UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Andrej Kocen

Razvoj 3D iger za uporabo na pametnih telefonih in tabličnih računalnikih

DIPLOMSKO DELO NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Franc Jager

Ljubljana, 2014

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01984 / 2014
Datum: 11.2.2014

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ANDREJ KOCEN**

Naslov: **RAZVOJ 3D IGER ZA UPORABO NA PAMETNIH TELEFONIH IN TABLIČNIH RAČUNALNIKIH**
DEVELOPMENT OF 3D GAMES FOR THE USE ON SMART PHONES AND TABLET COMPUTERS

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

Z uporabo orodja Unity opišite metodologijo in pristope za razvoj 3D iger, ki delujejo na pametnih telefonih in na tabličnih računalnikih z operacijskim sistemom Android. Z orodjem razvijte pripadajoči uporabniški vmesnik in opišite glavne naloge razvoja 3D iger kot so: nadzor in upravljanje 3D elementov v prostoru, delo z zvokom, delo z 2D teksturami ter povezovanje vseh teh elementov v zaključeno celoto. Z uporabo orodja Unity razvijte tudi igro v kateri igralec z nagibanjem mobilne naprave krmili letalo in poizkuša prevoziti čim daljšo razdaljo.

Mentor: prof. dr. Franc Jager

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan: prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani **Andrej Kocen**,

z vpisno številko **63020078**,

sem avtor diplomskega dela z naslovom:

Razvoj 3D iger za uporabo na pametnih telefonih in tabličnih računalnikih

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. Dr. Franca Jagra
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Mežici, dne: 28.3.2014 Podpis avtorja:

Zahvala

Rad bi se zahvalil svojemu mentorju, prof. dr. Francu Jagru, za strokovno vodenje in pomoč pri izdelavi diplomskega dela.

Zahvaljujem se tudi svojim staršem in vsem ostalim družinskim članom za vso podporo in pomoč.

KAZALO VSEBINE

POVZETEK

Tema diplomskega dela je razvoj 3D igre, ki deluje na pametnih telefonih in tabličnih računalnikih z operacijskim sistemom Android. Moja naloga je bila izbrati ustrezna razvojna orodja in z njimi razviti igro, jo objaviti na distribucijskem portalu Google play in pokazati rezultate s tega portala.

Razvil sem igro, v kateri igralec prevzame vlogo pilota lovskega letala in poskuša prevoziti čim daljšo razdaljo. Igralec z nagibanjem mobilne naprave krmili letalo, z dotikom na zaslon pa izstreljuje rakete. Igralno polje se gradi dinamično, odvisno od trenutne pozicije letala. Za letalom pa se igralno polje odstranjuje, da se s tem sproščajo resursi naprave. Razvil sem tudi pripadajoči grafični uporabniški vmesnik z uporabo elementov, ki jih ponuja orodje Unity.

Tekom razvoja sem z različnih spletnih portalov zbiral primerne 3D modele, zvoke in teksture. Teksture sem uporabljal predvsem pri izgradnji grafičnega uporabniškega vmesnika. Po potrebi sem vse elemente prilagodil in jih s programiranjem povezal v delujočo celoto. Na koncu sem igro objavil na portalu Google play in s tem omogočil dostop do uporabnikov po celem svetu.

Za osrednje razvojno orodje sem izbral Unity, za programiranje sem uporabil MonoDevelop in programski jezik C#. Za delo z zvoki sem uporabljal program Audacity, za delo s teksturami program Paint.NET, za oblikovanje preprostih 3D modelov pa program Blender.

Ključne besede:

Unity, razvoj, programski jezik C#, grafični uporabniški vmesnik, 3D igra

ABSTRACT

The topic of this thesis is development of a 3D game which works on smart phones and on tablet computers which are running on operating system Android. My goal was to find the appropriate development tools, develop the game, publish it on the distribution portal Google play and display the results.

I developed a game in which the player takes a role of a fighter jet pilot and tries to achieve as high distance as possible. Player is steering the fighter jet with tilting the device, with touch of the screen he shoots rockets. Playing field is generated dynamically, depending on the current jet location. Behind the jet, playing field is removed to free the resources of the device. I also developed graphical user interface with usage of appropriate elements which are offered by Unity.

During the development I gathered 3D models, sounds and textures from different websites and imported them into the project. I used textures mostly for creating graphical user interface. If needed, I adjusted imported elements and with programming connected them into working project. At the end I published the game on Google play portal and made it accessible from all over the world.

For main development tool I used Unity, for programming I used MonoDevelop with programming language C#. For working with sounds I used tool Audacity and for working with textures I used Paint.NET. I used Blender for modeling simple 3D models.

Keywords:

Unity, development, programming language C#, graphical user interface, 3D game

1 UVOD

Razvil sem 3D igro, ki deluje na operacijskem sistemu Android. Prilagodil sem jo, da dela na pametnih telefonih in tabličnih računalnikih. V tej igri prevzame igralec vlogo pilota lovskega letala in skuša prevoziti čim daljšo razdaljo. Letalo usmerja z nagibanjem mobilne naprave, z dotikom na zaslon pa strelja rakete. Igra torej bere vrednosti iz senzorjev mobilne naprave, zato (še) ni primerna za Google TV.

Pred in med razvojem sem moral izvršiti izbor najbolj primernih razvojnih orodij za projekt. Za razvoj igre sem uporabil programsko orodje Unity, za programiranje pa MonoDevelop. Veliko večino programiranja sem opravil s programskim jezikom C#, nekaj pa tudi z JavaScript. Kompleksnejše 3D modele, kot so na primer letala, sem poiskal na spletnih portalih, preproste pa sem ustvaril sam, večinoma znotraj programa Unity, nekaj pa tudi s programom Blender. Igri sem dodal tudi različne zvoke (glasba v ozadju, eksplozije, ...), ki sem jih našel na internetnih portalih in jih po potrebi prilagodil. Z generatorji tekstur, ki so prosto dostopni na spletu sem ustvaril 2D teksture in jih po potrebi s programom Paint.net prilagodil. Igro sem tudi objavil na portalu Google Play, ter kasneje še na Apple iTunes.

Namen diplomskega dela je prikazati zgoraj opisani postopek razvoja, ter prikaz postopka objave na Google Play portalu in predstaviti tam dosežene rezultate. Na kratko bom tudi opisal vsa razvojna orodja, ki sem jih uporabljal tekom razvoja.

Operacijski sistem Android

Operacijski sistem Android je postal najbolj razširjen operacijski sistem za mobilne naprave. Število naprav s tem operacijskim sistemom raste zelo hitro. Nameščen je že na več kot milijardo mobilnih naprav, vsak dan se jih na novo aktivira več kot milijon. Razlog za priljubljenost in številčnost je predvsem odprtokodnost sistema, ki omogoča različnim proizvajalcem strojne opreme, da si ga prilagodijo po svojih potrebah. Na sliki 1 je prikazana primerjava razširjenosti štirih najpopularnejših operacijskih sistemov.

Top Four Operating Systems, Shipments, and Market Share, Q3 2013 (Units in Millions)

Operating System	3Q13 Shipment Volumes	3Q13 Market Share	3Q12 Shipment Volumes	3Q12 Market Share	Year-Over-Year Change
Android	211.6	81.0%	139.9	74.9%	51.3%
iOS	33.8	12.9%	26.9	14.4%	25.6%
Windows Phone	9.5	3.6%	3.7	2.0%	156.0%
BlackBerry	4.5	1.7%	7.7	4.1%	-41.6%
Others	1.7	0.6%	8.4	4.5%	-80.1%
Total	**261.1**	**100.0%**	**186.7**	**100.0%**	**39.9%**

Source: IDC Worldwide Mobile Phone Tracker, November 12, 2013

Slika 1: Primerjava razširjenosti najpogostejših operacijskih sistemov za mobilne naprave za obdobje tretjega četrtletja leta 2012 in 2013. V zadnjem stolpcu je v odstotkih prikazana rast, oziroma padec števila naprav.

Google Play

Največji portal za distribucijo aplikacij za operacijski sistem Android se imenuje Google Play. Z njim upravlja podjetje Google, javnosti je bil prvič dostopen 23. oktobra 2008. Od takrat naprej število objavljenih aplikacij in število inštalacij s tega portala strmo narašča. Leta 2013 je število nameščenih aplikacij preseglo 50 milijard, na portalu pa je naloženih že več kot milijon aplikacij [1].

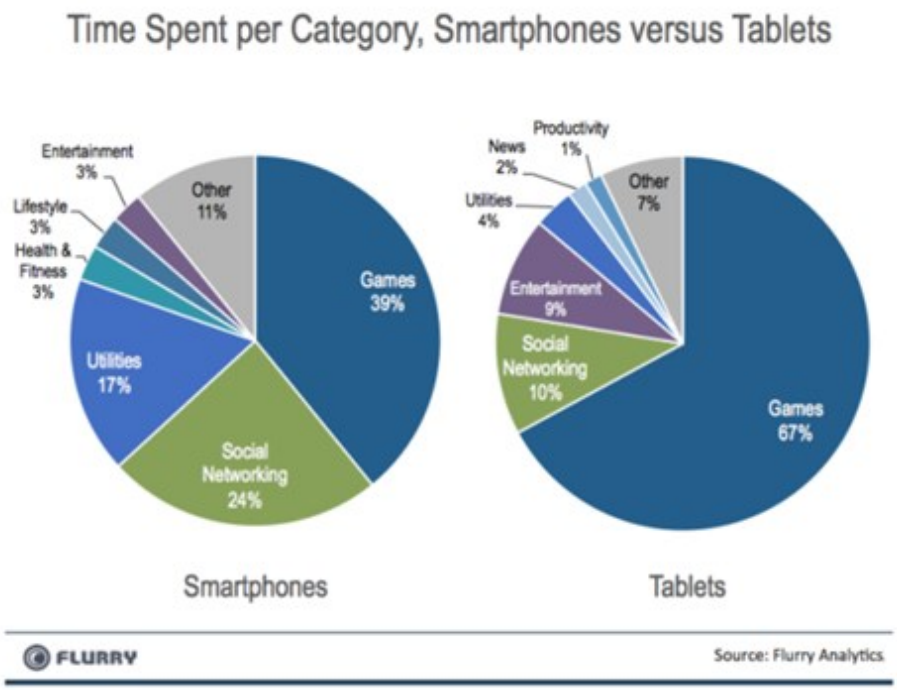
Razvoj igre

Pri razvoju 3D iger naletimo na mnogo različnih nalog, kot so delo z zvokom, modeliranje v 3D prostoru in delo s teksturami. S programiranjem pa vse te elemente povežemo skupaj v delujočo celoto, ter določimo obnašanje 3D elementov v prostoru.

Rezultat razvoja je 3D igra, ki deluje na operacijskem sistemu Android. Naslednji korak pa je objava te igre na različnih distribucijskih portalih. Podrobneje bom opisal Google play portal, prikazal odpiranje Google play razvijalskega računa in nalaganje razvite igre na ta portal. Opisal bom tudi rezultate, odzive uporabnikov in nadaljnje načrte za razvoj igre.

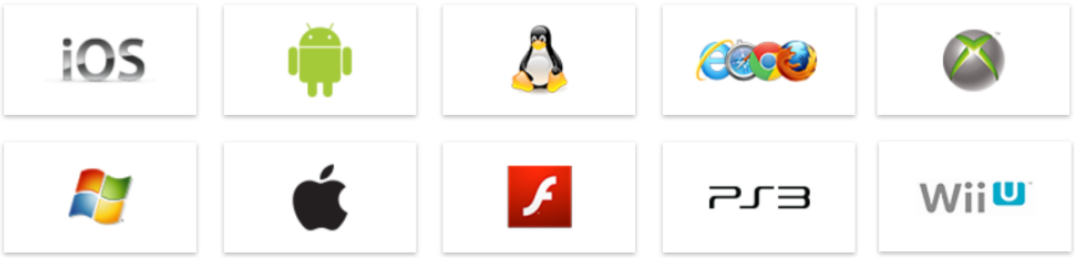
2 OPIS PROBLEMA

Največji delež aplikacij na platformi Android predstavljajo igre. Raziskave kažejo tudi, da uporabniki porabijo največ časa za njih (Slika 2) in manj za ostale tipe aplikacij. S hitrim razvojem tehnike pa je omogočen razvoj procesorsko zahtevnih iger tudi v 3D prostoru. Take igre so delovale še nekaj let nazaj le na prenosnih in na stacionarnih računalnikih, ter na igralnih konzolah.



Slika 2: Porazdelitev časa, ki ga uporabnik porabi za različne tipe aplikacij in primerjava te porazdelitve na pametnih telefonih in na tabličnih računalnikih. Statistika je pridobljena z več kot 6 milijard aplikacijskih sej in na približno 500 milijonih pametnih telefonov in tabličnih računalnikov.

Za razvoj 3D igre potrebujemo 3D grafični pogon. Trenutno sta za mobilne platforme najbolj priljubljena Unreal engine in Unity. Za grafični pogon je pomembno, da nam čim bolj poenostavi in pohitri proces razvoja igre. Vsebovati mora pregleden uporabniški vmesnik, učinkovit fizikalni pogon (Unity ima vgrajen NVIDIA PhysX fizikalni pogon), modul za delo z zvokom... Pomembno je tudi, da nam omogoča enostaven prehod na druge operacijske sisteme, na primer iOS. Unity poleg operacijskega sistema Android in iOS podpira še mnogo drugih operacijskih sistemov (Slika 3). Zaradi enostavnejše pridobitve licence sem se odločil za Unity basic licenco z dodatkom za platformo Android. Unity basic licenca je brezplačna.



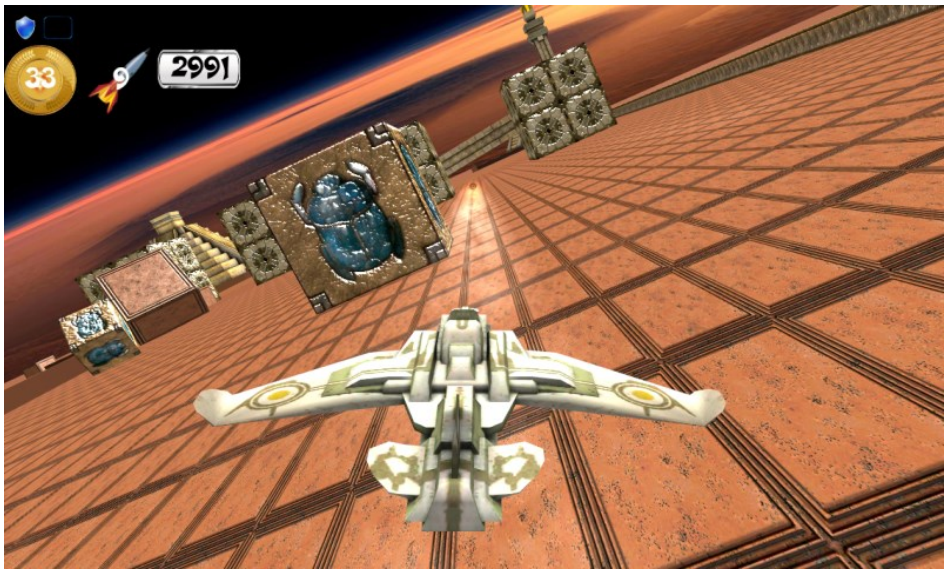
Slika 3: Unity omogoča razvoj na naslednjih platformah: iOS, Android, Linux, Web player (Internet explorer, Firefox, ...), Xbox360, Windows, Mac, Flash, Play Station 3, Wii U.

2.1 Opis igre

Igra, ki sem jo razvil, spada v kategorijo t.i. "Endless runner" iger. Značilno za njih je, da se igralno polje gradi sproti, medtem ko igralec napreduje. Teoretično bi tako lahko igralec igral neskončno dolgo, kar pove tudi samo ime te zvrsti. Cilj pa je doseči čim več točk. Najbolj znan predstavnik te zvrsti je trenutno igra Temple run, ki je že presegla 100 milijonov namestitev (skupno na Android in iOS platformi).

Igralec v moji igri prevzame vlogo pilota lovskega letala in poskuša prevoziti čim daljšo razdaljo. Na njegovi poti se pojavljajo številne ovire, ki ogrožajo potovanje. Na poti se naključno generirajo tudi predmeti, ki so igralcu v pomoč - nagrade (npr. dodatne rakete). Težavnost se s časom povečuje in ovire postajajo vse bolj zahtevne. S časom se povečuje tudi hitrost letala in s tem se še dodatno povečuje težavnost. Na začetku ima igralec na voljo eno letalo, na katerem lahko nadgrajuje ščit, število raket, število raketometov, odklene lahko tudi laserski namerilec. Med igranjem si lahko prisluži denar v obliki zlatnikov (Slika 4) in kasneje kupi boljše letalo, ki ga prav tako lahko nadgrajuje. Število nadgradenj je omejeno in odvisno od posameznega letala. Poleg nadgrajevanja letal je možno nadgrajevati tudi same nagrade (poglavje 4.1), ki jih igralec pobira med igranjem. Igralec kontrolira letalo z nagibanjem naprave ter strelja rakete (Slika 5) z dotikom na zaslon.

Slika 4: Posnetek med igranjem. V zgornjem levem kotu je prikazana moč ščita, število pobranih zlatnikov, število raket in dosežena razdalja. Na levi strani so vidni zlatniki, ki jih je možno pobrati.



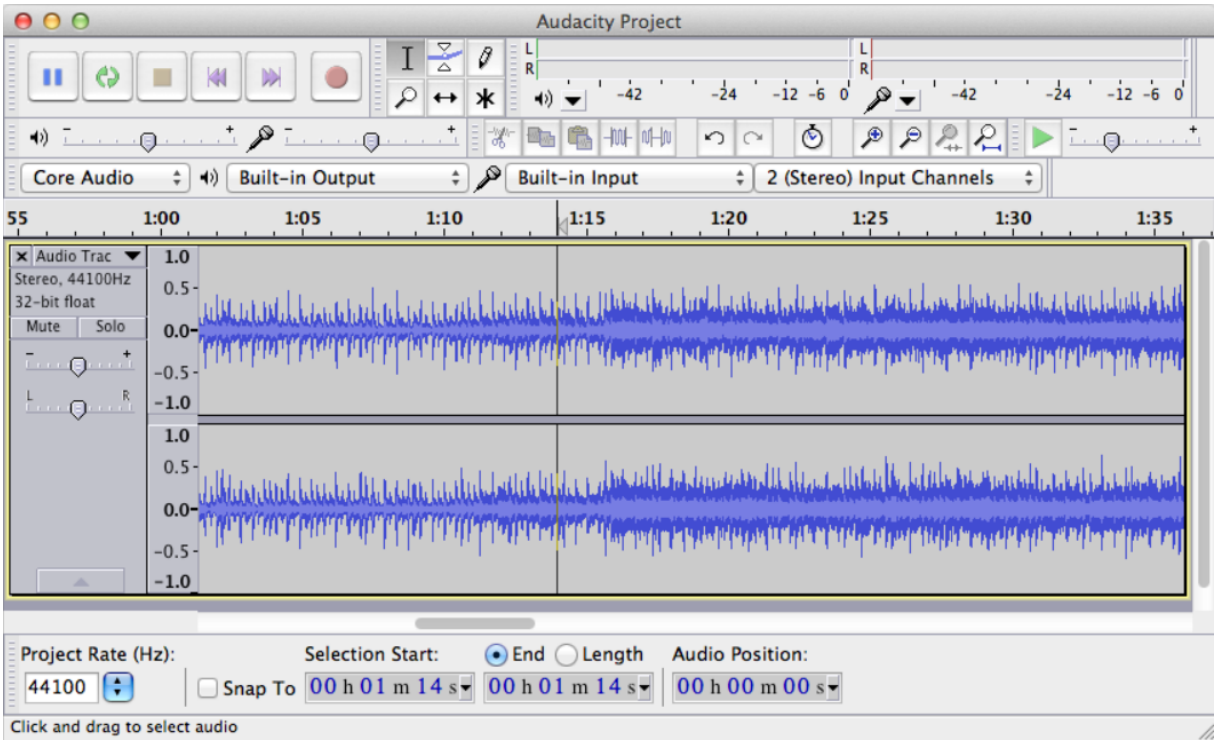
Slika 5: Izstrelitev rakete.

3 OPIS UPORABLJENIH RAZVOJNIH ORODIJ

V tem poglavju bom zelo na kratko opisal orodja, ki sem jih uporabljal pri razvoju igre. Največji poudarek bo seveda na orodju Unity.

3.1 Audacity

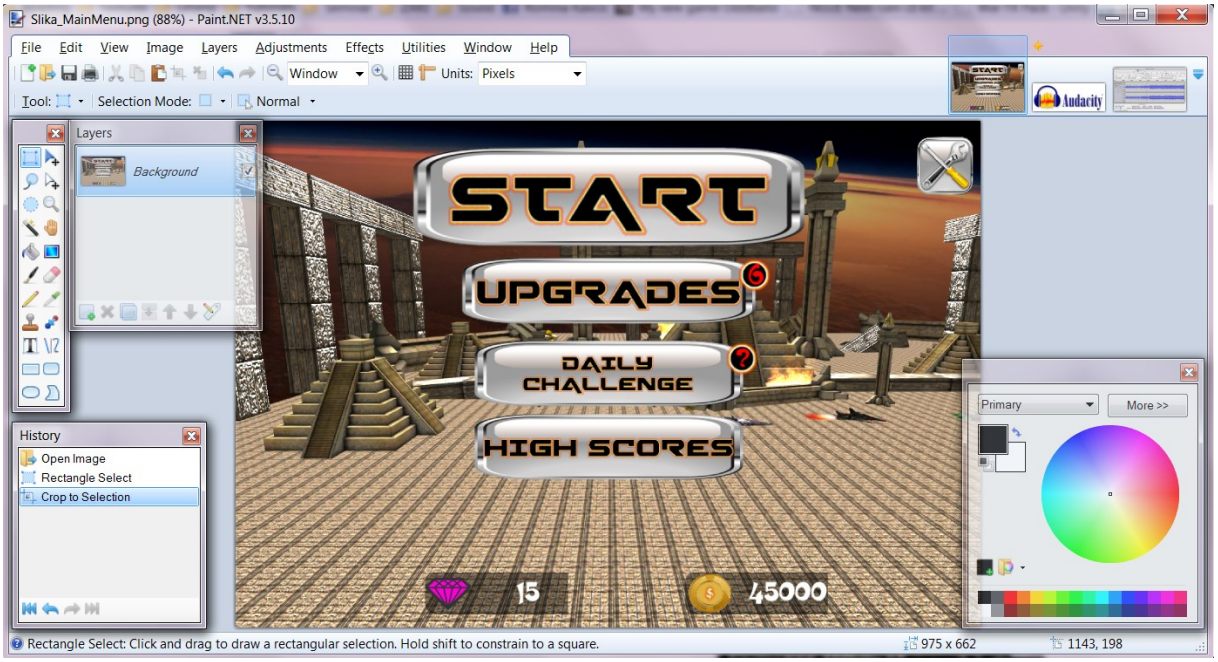
Je brezplačen odprtokodni program za delo z zvoki[2] (Slika 6). Deluje na operacijskih sistemih Windows, Mac OS X in GNU/Linux. Deluje nad zvokovnimi formati WAV, AIFF, FLAC, MP2, MP3 or Ogg Vorbis. V svojem projektu sem ga uporabljal predvsem za kompresijo, rezanje zvokov ter pretvorbo med različnimi zvokovnimi formati.



Slika 6: Uporabniški vmesnik programa Audacity.

3.2 Paint.NET

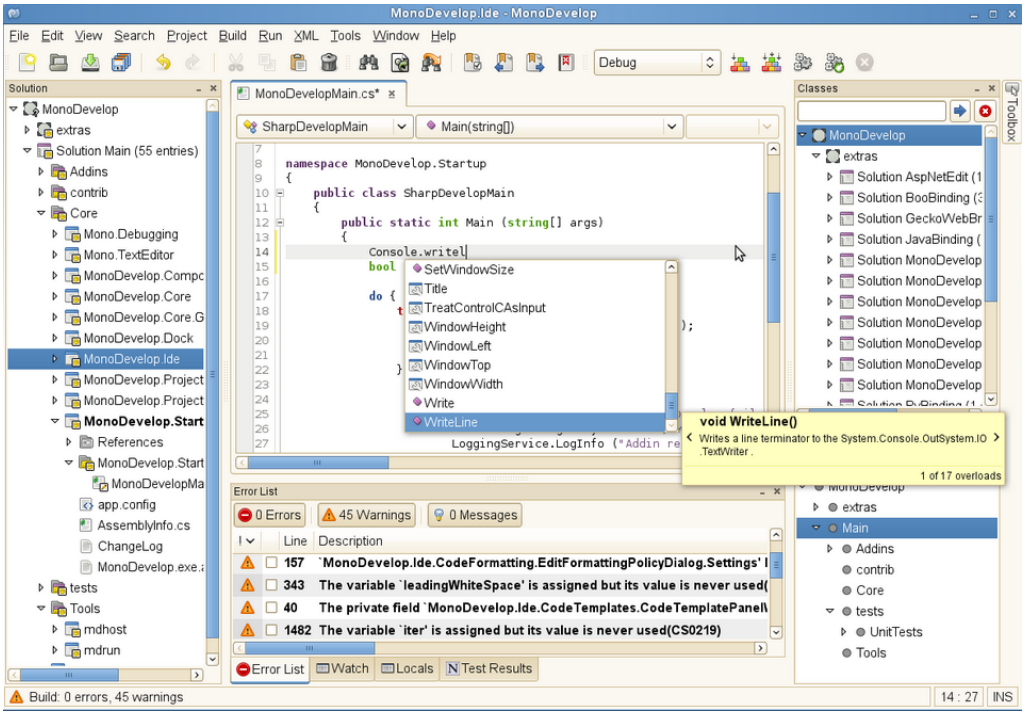
Je brezplačen program (Slika 7) za delo s teksturami [3]. Deluje na operacijskem sistemu Windows. Omogoča delo nad formati PNG, JPEG, BMP, GIF, TGA, DDS in TIFF. Pri mojem projektu sem ga največkrat uporabljal za spreminjanje velikosti tekstur in za pripravo elementov za GUI.



Slika 7: Uporabniški vmesnik programa Paint.NET.

3.3 MonoDevelop

Je brezplačno razvojno okolje (Slika 8), namenjeno programiranju v programskih jezikih C#, JavaScript, Visual Basic, C, C++, Boo in Vala. Deluje na operacijskih sistemih Windows, Linux in MAC OS X. V svojem projektu sem uporabljal MonoDevelop za vso programiranje [4].



Slika 8: Uporabniški vmesnik programa MonoDevelop.

3.4 Blender

Blender je brezplačen program, namenjen 3D oblikovanju [5]. Z njim je mogoče izdelovati od preprostih 3D modelov, pa do zelo kompleksnih. Deluje na operacijskih sistemih Windows, Mac OS X, Linux, ter FreeBSD. V svojem projektu sem uporabil Blender za oblikovanje preprostih modelov, na primer, za oblikovanje magneta (Slika 36).

3.5 Unity

Programsko orodje Unity se najpogosteje uporablja za izdelavo 3D iger, v manjši meri tudi za izdelavo 2D iger in za izdelavo simulacij v 3D prostoru.

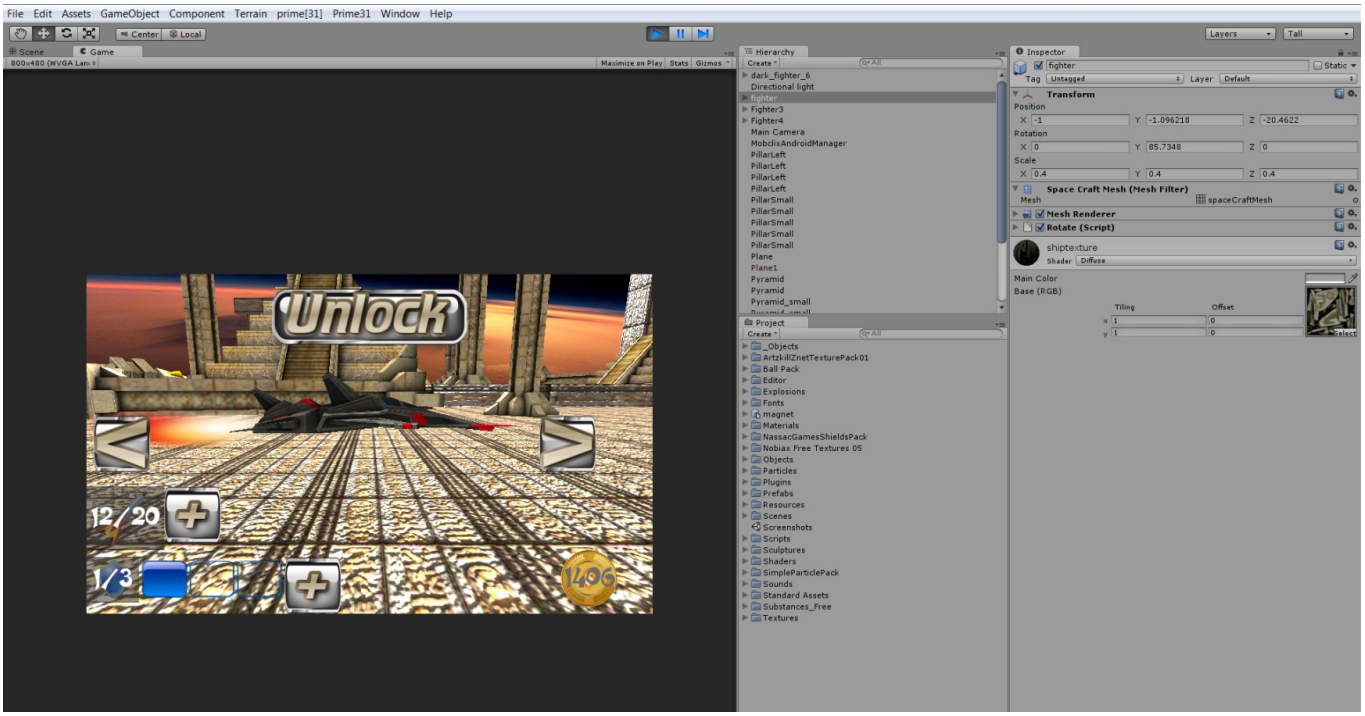
Na samem začetku razvoja imamo prazen prostor v katerega postavljamo objekte (v okolju Unity se imenujejo GameObjects). Vsak objekt je sestavljen iz najmanj ene komponente, imenovane Transform (Slika 10). Transform natančno določa položaj, rotacijo in velikost objekta. Vsi trije našteti elementi so opisani s koordinatami x, y in z. Komponento Tranform vsebuje vsak objekt, tudi prazen.

Razvoj v okolju Unity je kombinacija dela v samem urejevalniku in programiranja. V urejevalnik uvažamo različne tipe elementov (3D modeli, zvoki, teksture), jih postavljamo v igralni prostor, sestavljamo jih lahko v razmerja potomec - starš in jim dodajamo različne komponente (poglavje 3.5.4). S programiranjem pa natančno določimo njihovo delovanje.

3.5.1 Uporabniški vmesnik Unity

Uporabinški vmesnik (Slika 9) je v osnovi razdeljen na štiri glavne dele:

- Igralni prostor (v okolju Unity se imenuje Scene, oziroma Game): nam prikazuje postavitev objektov in omogoča predogled in testiranje igre. Vanj vnašamo nove objekte in določamo njihovo lokacijo, rotacijo in velikost. Na ta način v bistvu gradimo igralni prostor.
- Prikaz hierarhije: tukaj vidimo vse objekte, ki se nahajajo v igralnem prostoru. Če tukaj izberemo en objekt, se njegove podrobnosti prikažejo v panelu inspektor.
- Panel inspektor: prikazuje podrobnosti posameznega objekta. Objekt, katerega podrobnosti želimo videti lahko izberemo v panelu hierarhije ali pa v panelu igralni prostor.
- Panel projekt: prikazuje vse elemente, ki so v projektu. Posamezen projekt lahko vsebuje več igralnih prostorov. Vsebuje lahko tudi elemente, ki se v nobenem igralnem prostoru sploh ne pojavljajo.



Slika 9: Uporabniški vmesnik razvojnega orodja Unity.

3.5.2 Igralni prostor in igralno polje

Na igralni prostor lahko gledamo kot na posamezno sobo v igri. Vanj spada vse, kar lahko igralec vidi, tudi elemente, katerim se ne more nikoli približati (na primer nebo) in tudi elemente GUI. Z vidika razvijalca je na začetku viden kot neskončno velik prazen prostor, v katerega postavlja objekte. Znotraj igralnega prostora mora razvijalec določiti igralno polje.

Igralno polje je področje, kjer se lahko igralec giblje. Vsaka igra na nek način omejuje igralčevo gibanje, najpogosteje so to zidovi ali pa tudi strme gore, ki jih igralec ne more prečkati. V svoji igri določam igralno polje z zidovi, ki omejujejo gibanje v levo in desno stran.

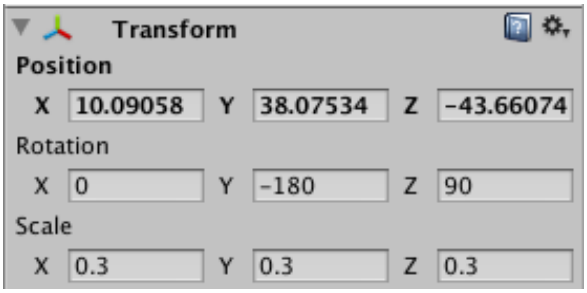
3.5.3 Objekti v okolju Unity (GameObjects)

So temeljni elementi Unity pogona, iz njih je sestavljen celoten igralni prostor. Sestavljeni so iz komponent (poglavje 3.5.4), ki določajo njihove lastnosti. Objekti so lahko preprosti (prazen objekt) ali pa zelo kompleksni, na njih lahko delujejo fizikalne sile, ki so vgrajene v Unity pogon in/ali jih upravljamo s programiranjem preko skript. Objekte lahko povezujemo med seboj v relacije starš – potomec - enakovreden nivo.

V naslednjih podpoglavjih bom opisal nekaj glavnih tipov objektov v okolju Unity [7].

3.5.3.1 Prazen objekt

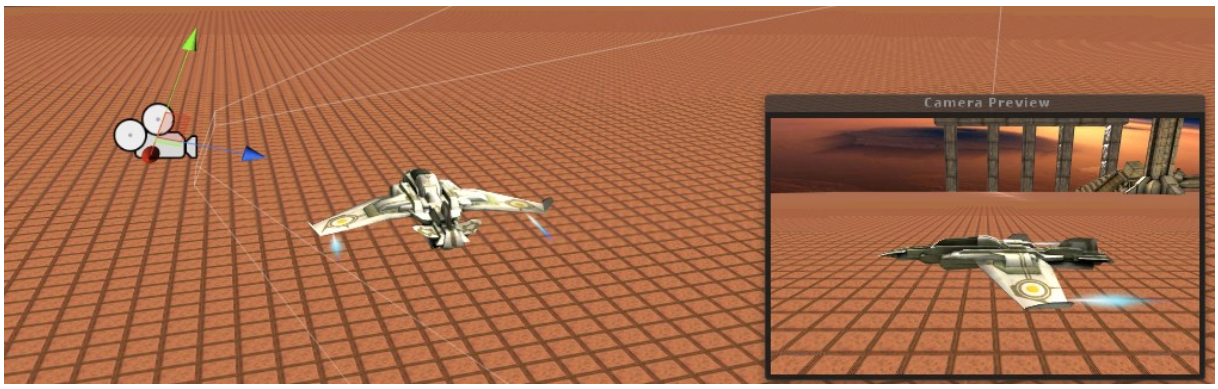
Prazen objekt v okolju Unity je vsak objekt, ki vsebuje le komponento Transform (Slika 10) in s tem opis položaja, rotacije in velikosti. Ne vsebuje pa nobenih drugih komponent. Uporaben je predvsem zato, da na njega vežemo skripte.



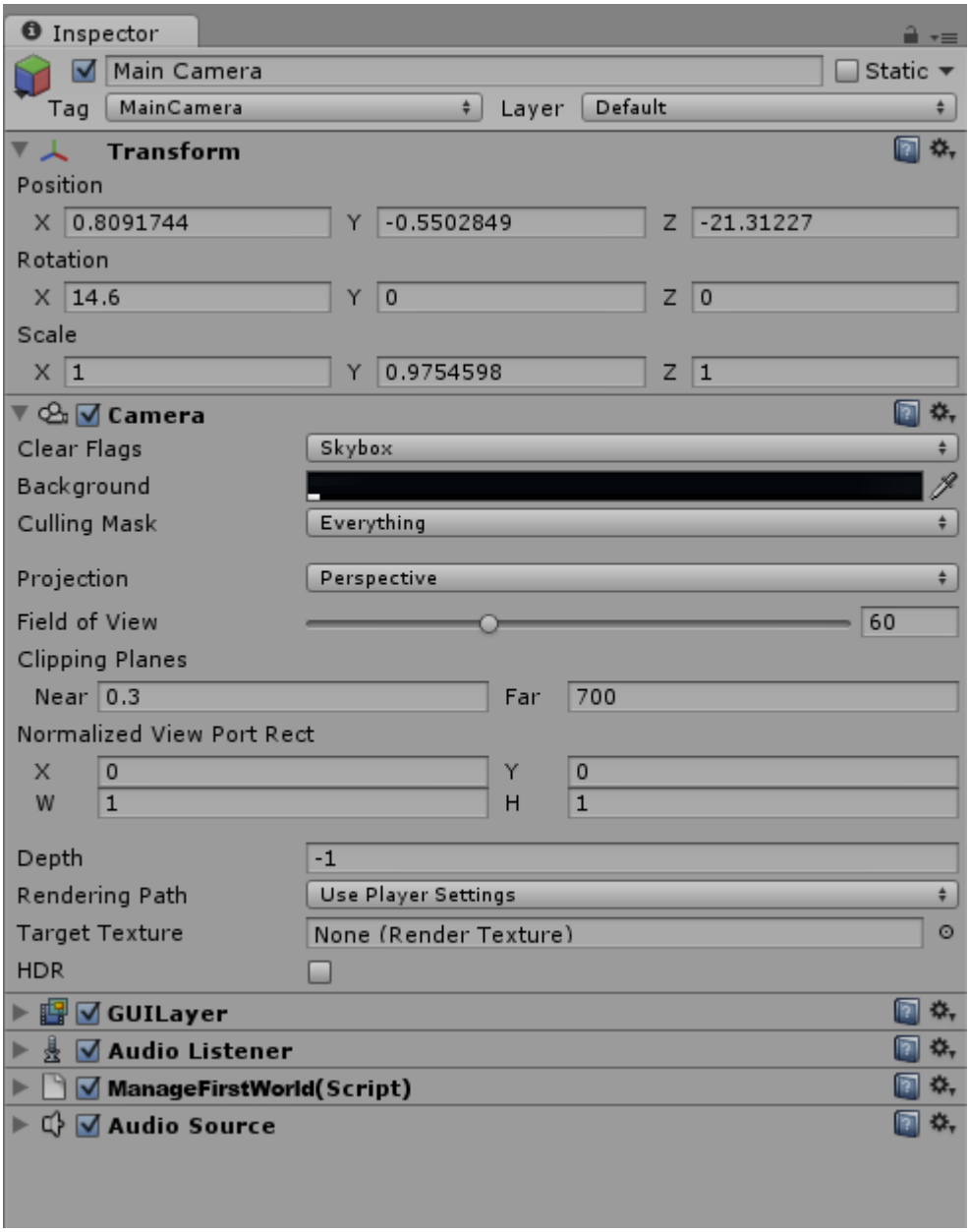
Slika 10: Komponenta Transform določa lokacijo, rotacijo in velikost oziroma razmerje velikosti, glede na začetno velikost 3D objekta. Vsak 3D objekt ima namreč na začetku (ob uvozu v Unity) razmerje velikosti enako ena, po vseh oseh.

3.5.3.2. Kamera

Da sploh lahko karkoli vidimo, moramo v igralni prostor postaviti kamero. Ta predstavlja naše oči. S premikanjem in rotacijo kamere po prostoru dosežemo vtis, kot da se tudi mi premikamo po prostoru. V prostoru lahko imamo več kamer hkrati, če želimo opazovati dogajanje iz več kotov. Na sliki 11 je prikazan primer postavitve kamere, na sliki 12 pa njene nastavitve.



Slika 11: Primer postavitve kamere in prikaz predogleda skozi kamero.



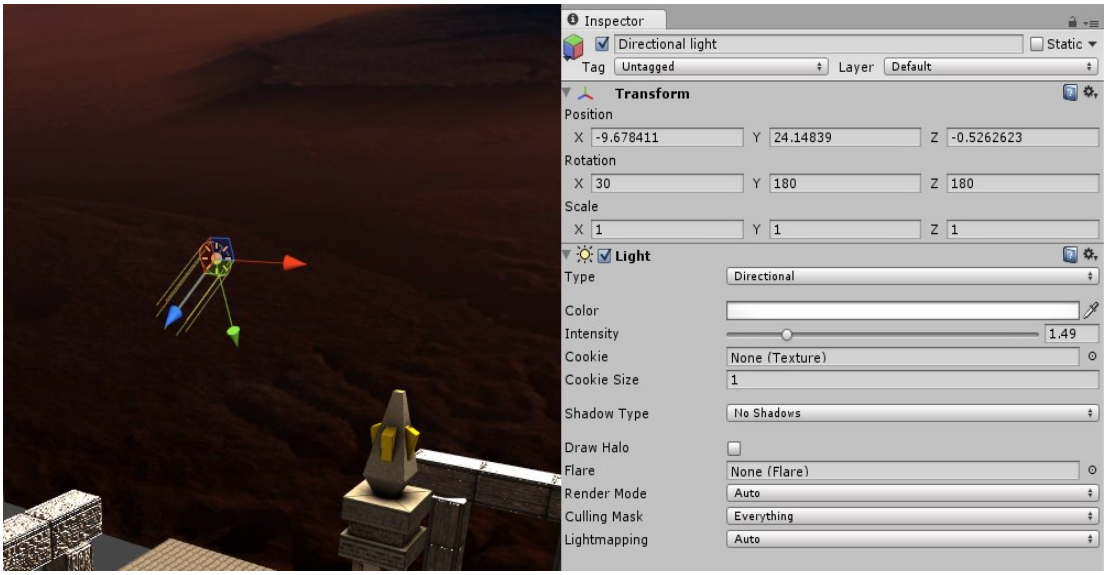
Slika 12: Primer nastavitve kamere. Na kamero so pritrjene še komponente GUILayer, Audio Listener, skripta ManageFirstWorld.cs in Audio Source.

3.5.3.3 Viri svetlobe

Ko je v prostoru postavljena kamera, je naslednji korak postavitev virov svetlobe. Brez njih je igralni prostor temen. Z njimi osvetljujemo prostor in simuliramo različne tipe virov svetlobe iz realnega sveta. Simuliramo lahko od sonca do npr. vžigalice, odvisno od nastavitev in tipa vira svetlobe. Spodaj so opisani osnovni trije tipi virov svetlobe.

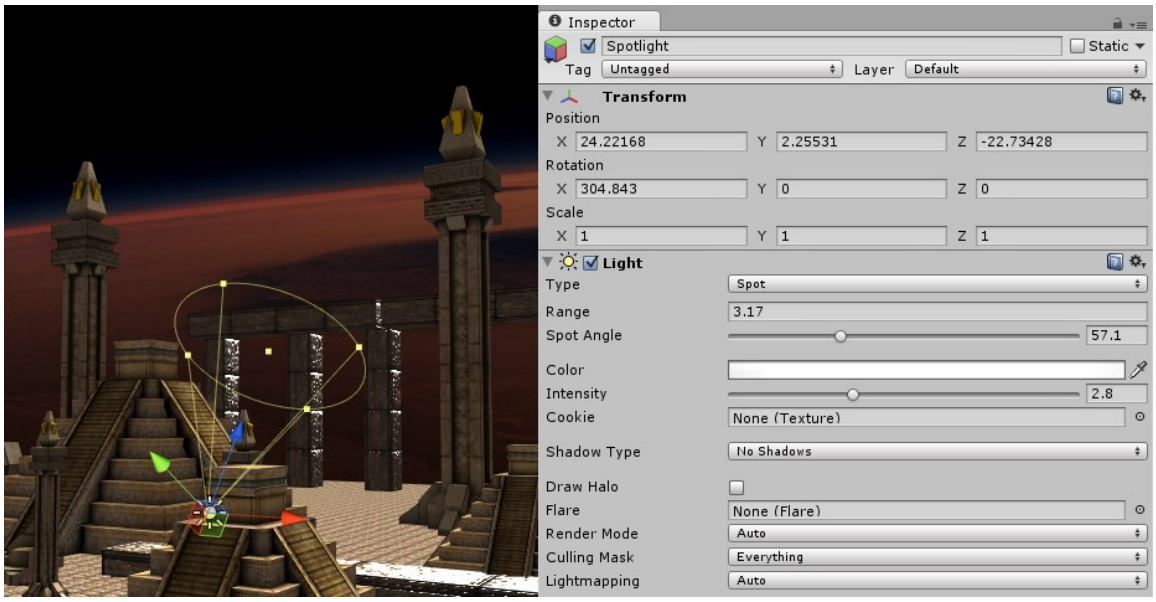
- Usmerjen vir svetlobe: je lahko postavljen "neskončno daleč" od objektov. Osvetljuje vse objekte v igralnem prostoru enako, njegova jakost z oddaljenostjo ne slabi.

Najpogosteje se uporabljajo za simulacijo Sonca ali Lune. Njegove najpomembnejše nastavitve so: jakost, barva ter usmerjenost svetlobe (Slika 13).



Slika 13: Usmerjen vir svetlobe in njegove nastavitve v prvem igralnem prostoru.

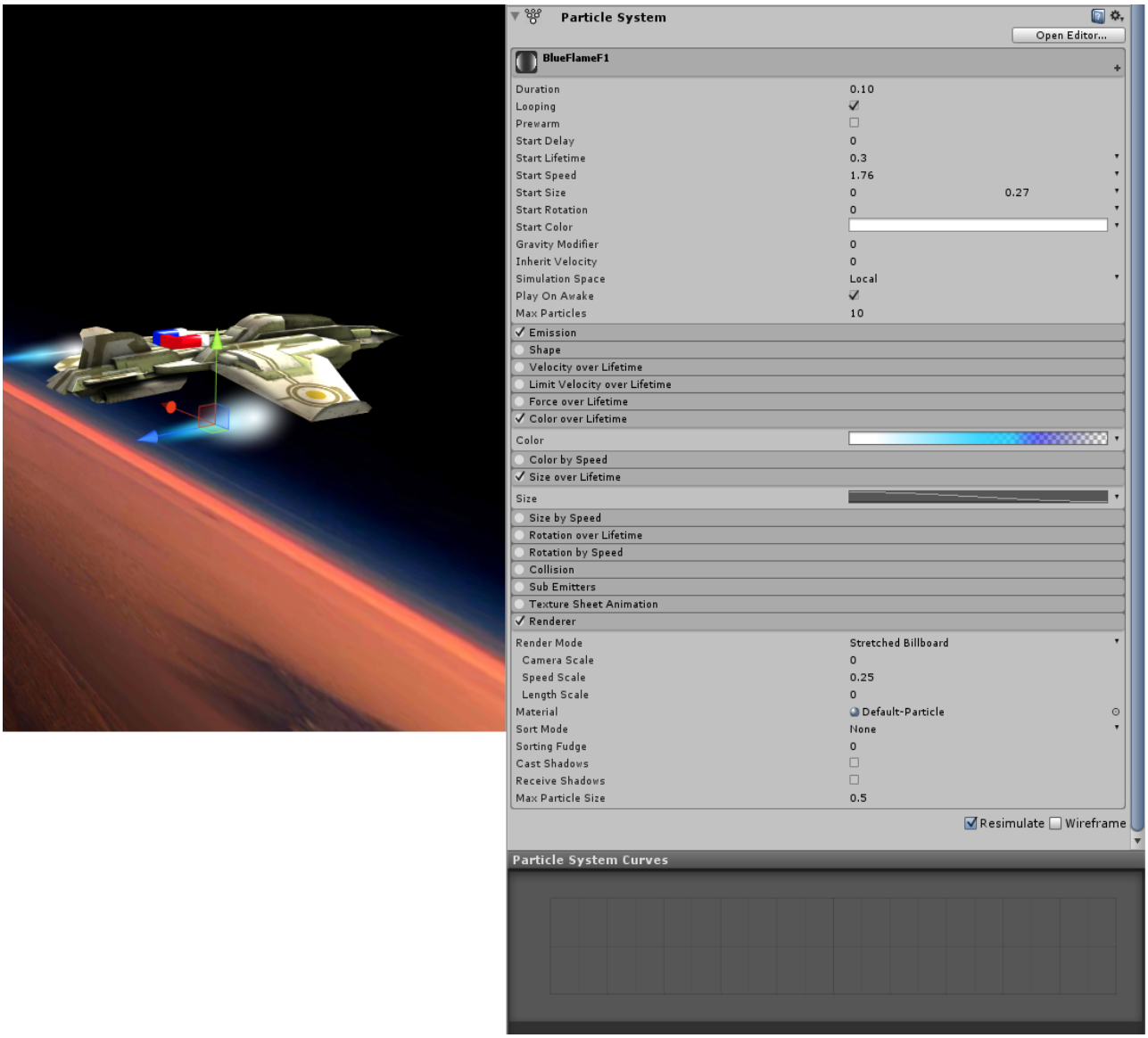
- Točkovni vir svetlobe: oddaja svetlobo iz ene točke v vse smeri enakomerno. Svetlobo oddaja iz določene točke, le do razdalje, ki jo določimo v nastavitvah. Ti viri svetlobe se najpogosteje uporabljajo za simulacijo eksplozij, žarnic in podobno. Jaz teh virov svetlobe v svoji igri zaradi optimizacije ne uporabljam.
- Usmerjen točkovni vir svetlobe (Slika 14): podobno kot točkovni vir svetlobe, oddaja ta vir svetlobo iz določene točke, le do določene razdalje. Vendar pa je ne oddaja v vse smeri, ampak jo oddaja v obliki stožca le v določeno smer. Ta tip svetlobe se najpogosteje uporablja simulacijo ročnih svetilk, avtomobilskih žarometov in podobno.



Slika 14: Usmerjen točkovni vir svetlobe z nastavitvami v prvem igralnem prostoru.

3.5.3.4 Sistem za simulacijo delcev

Najpogosteje se uporabljajo za simulacijo ognja, dima, eksplozij, oblakov in podobno. V svoji igri jih uporabljam za simulacijo eksplozij, za simulacijo dima, ki nastaja za izstreljeno raketo in za simulacijo izpušnega ognja pogona letala (Slika 15).



Slika 15: Primer sistema za simulacijo delcev. Simulira izpušni ogenj iz motorja letala. Na desni strani slike so vidne tudi nastavitve za ta sistem.

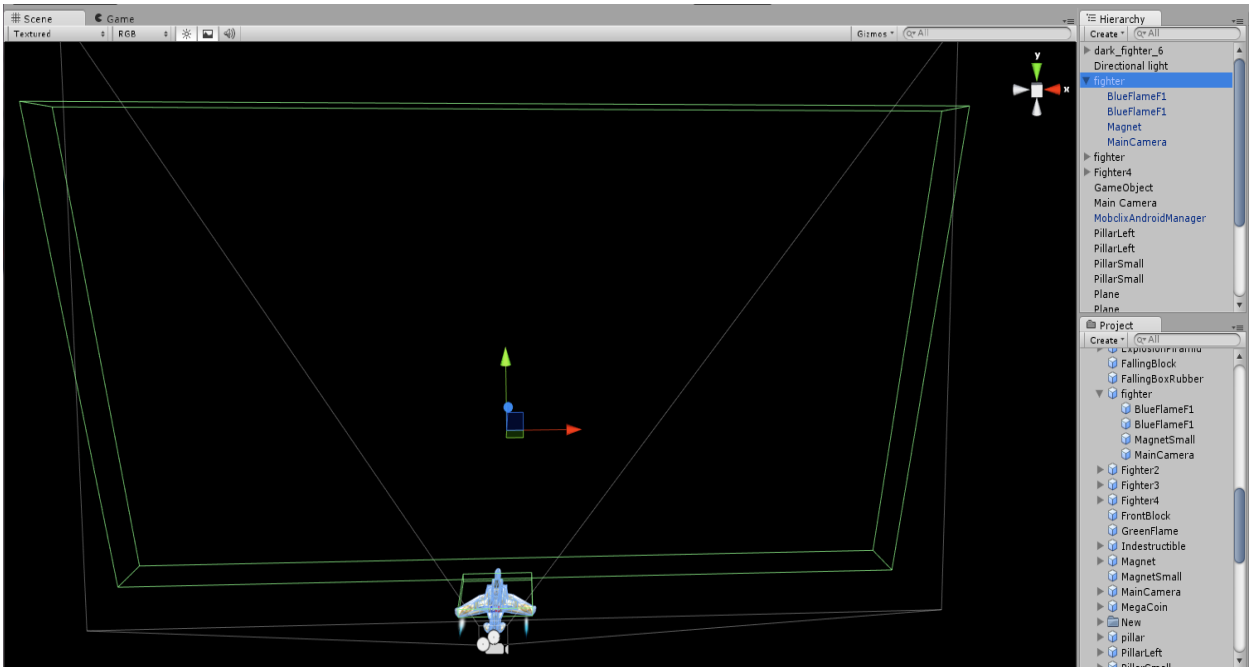
3.5.3.5 Sestavljeni objekti

Lahko so sestavljeni iz dveh ali več osnovnih objektov. Sestavljene objekte gradimo po modelu starš - potomec - enakovreden nivo (ang. parent - child - sibling). Pomembno je vedeti, da se vsaka sprememba položaja, rotacije ali velikosti samodejno prenaša na vse potomce. Uporaba sestavljenih objektov nam zato bistveno poenostavi razvoj. Primer

sestavljenega objekta je na primer letalo (Slika 15). Sestavljeno je iz naslednjih osnovnih objektov: letalo, magnet, kamera, dva sistema za simulacijo delcev.

3.5.3.6 Montažni objekti

Montažni objekt se v okolju Unity imenuje *prefab*. Uporabljamo jih predvsem kot objekte, ki se v igralnem polju v enaki obliki pojavljajo večkrat. Njihov namen je poenostaviti razvoj v tem smislu, da samo enkrat sestavimo nek objekt (ta objekt je lahko preprost ali pa sestavljen iz več kompleksnejših objektov), ga shranimo kot montažni objekt, potem pa ga lahko z drag&drop tehniko uporabimo neomejeno krat. Na tak način ustvarjamo instance tega objekta v igralnem polju. Če pa naredimo na njem neko spremembo, se ta sprememba prenese na vse instance v igralnem polju, zato nam ni potrebno delati popravkov na instancah. Pri razvoju igre sem delal praktično vse preko montažnih objektov. Primer so tudi vsa letala (Slika 16).



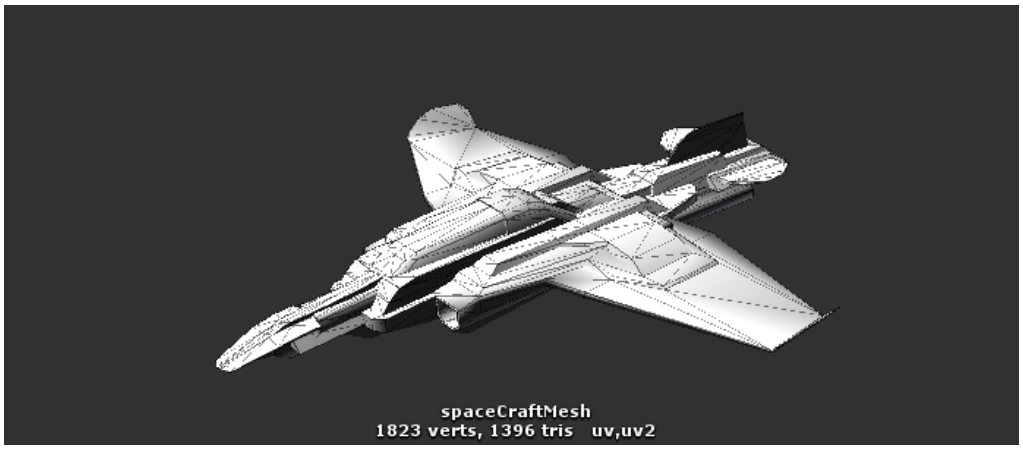
Slika 16: Primer montažnega objekta. Lovsko letalo je sestavljeno iz štirih osnovnih objektov: starševski objekt je letalo, potomci pa so: glavna kamera, magnet in dva sistema za simulacijo delcev (simulirata modri plamen pod krilih letala). Vsi ti objekti skupaj predstavljajo montažni objekt.

3.5.4 Komponente osnovnih modelov

To so komponente, ki jih dodajamo na osnovne objekte in jim na ta način dodajamo nove lastnosti. Z njimi lahko na primer določimo, ali bo na objekt delovala gravitacija, ali je objekt sposoben oddajati zvoke, ali naj fizikalni pogon zaznava trke objekta z drugimi objekti...

Komponente so vidne v panelu Inspector. Tipi komponent se med seboj po svojih lastnostih zelo razlikujejo. Najpogostejši so:

- Transform (Slika 10): opisuje lokacijo, rotacijo in velikost objekta. To komponento mora obvezno imeti vsak objekt. Če vsebuje le to komponento, pravimo da je objekt prazen.
- 3D mreža: predstavlja osnovni 3D model (Slika 17).



Slika 17: Mreža 3D modela letala.

- Izrisovalec mreže: brez njega je 3D model neviden, četudi obstaja v prostoru.
- Skripte: to so tekstovne datoteke, ki vsebujejo kodo. V orodju Unity je mogoče programirati v programskih jezikih C#, Javascript in Boo. V posameznem projektu lahko uporabljamo vse te jezike hkrati. V projektu za diplomsko nalogo sem uporabljal v veliki večini programski jezik C#. Preko skript s programiranjem kontroliramo osnovne objekte (poglavje 3.5.5).
- "Togo telo" (ang. Rigidbody): omogoča delovanje fizike na osnovni objekt. Preko njega lahko določamo maso, gravitacijo, upor pri premikanju objekta in podobno. Ta komponenta je nujna, če želimo da na objekt deluje fizikalni pogon NVIDIA PhysX.
- Trkalnik: Omogoča zaznavanje trkov med objekti. Če je objekt brez te komponente, zaznavanje trkov ni možno, v tem primeru se objekt neovirano giblje skozi vse ostale objekte.
- Vir zvoka: predstavlja točko v prostoru, kjer lahko nastajajo zvoki. V osnovi se zvoki delijo v 2D in 3D zvoke. 2D zvoki ohranjajo svojo jakost skozi celoten igralni prostor, 3D zvokom pa jakost pada z oddaljenostjo. Če je v igralnem prostoru vir zvoka, potem mora biti tudi vsaj en sprejemnik zvoka, ki je največkrat pritrjen na kamero.

- Sprejemnik zvoka: predstavlja ušesa igralca med igro. Navadno je pritrjen na kamero in sprejema vse zvoke, ki nastajajo v igralnem prostoru.

3.5.5 Programiranje v okolju Unity

S programiranjem vodimo dogajanje in povežemo posamezne dele igre v celoto. Z njim na primer ob določenih dogodkih prožimo zvoke, premikamo objekte, urejamo GUI...

Unity omogoča programiranje v treh programskih jezikih: C#, JavaScript in programski jezik Boo. Za programerska orodja si lahko izberemo Visual Studio ali MonoDevelop. Zanimivo pri tem je, da lahko znotraj istega projekta uporabljamo različne programske jezike. Pri izdelavi svoje igre sem večinoma uporabljal programski jezik C#.

Koda v projektu se v osnovi deli na posamezne skripte. To so datoteke, ki vsebujejo zaključene bloke kode. Te datoteke pritrdimo na objekte in jih na ta način vodimo oziroma kontroliramo. Na vsak objekt je možno pritrditi več skript.

```
using UnityEngine;

public class EmptyScript : MonoBehaviour
{
        /* Spodaj so na kratko opisane najbolj pogosto uporabljane
        funkcije, ki izhajajo iz objekta MonoBehaviour. Preko tega
        objekta se navezujemo na notranje delovanje Unity pogona. */

        /* Funkcija Start() se pokliče le enkrat na začetku izvajanja, še
        preden se prvič pokliče Update(). */
        void Start ()
        {
        }

        /* Funkcija Update() se kliče vedno pred izrisom slike na
        zaslon(število klicev na sekundo je torej odvisno od števila slik
        na sekundo, ki jih omogoča mobilna naprava). */
        void Update ()
        {
        }

        /* Funkcija OnGui() določa grafični uporabniški vmesnik(gumbi,
        labele, ...)omogoča izris sporočil in statistik, ravnotako pa
        poskrbi za sprejem različnih  informacij od uporabnika. */
        void OnGui()
        {
        }
}
```

Primer 1: Osnovne funkcije v skriptah.

4 RAZVOJ IGRE

Igro sem v osnovi razdelil na dva igralna prostora. V prvem igralnem prostoru lahko igralec izbira letalo, pregleduje najboljše rezultate, spreminja nastavitve - skratka v tem igralnem prostoru si igralec pripravi pogoje za igranje.

V drugem igralnem prostoru poteka dejansko igranje igre.

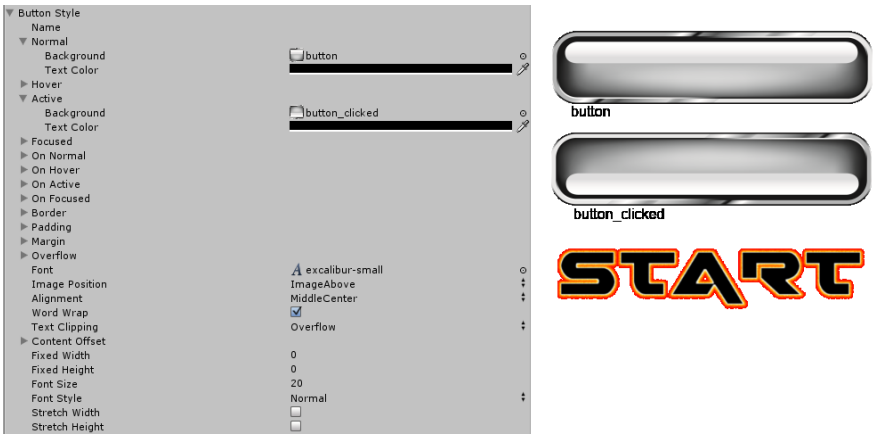
Izdelava elementov GUI

Vsi elementi GUI so vizualno prilagojeni k celostnemu izgledu igre. Za vsak tip elementa GUI (npr. gumb, labela) imam posebej oblikovan vizualni izgled oziroma stil, v urejevalniku Unity se imenuje GUI Style [15]. V njem je natančno določen izgled objektov ob vseh možnih dogodkih, oziroma stanjih.

Za vsak GUI Style je potrebno določiti:

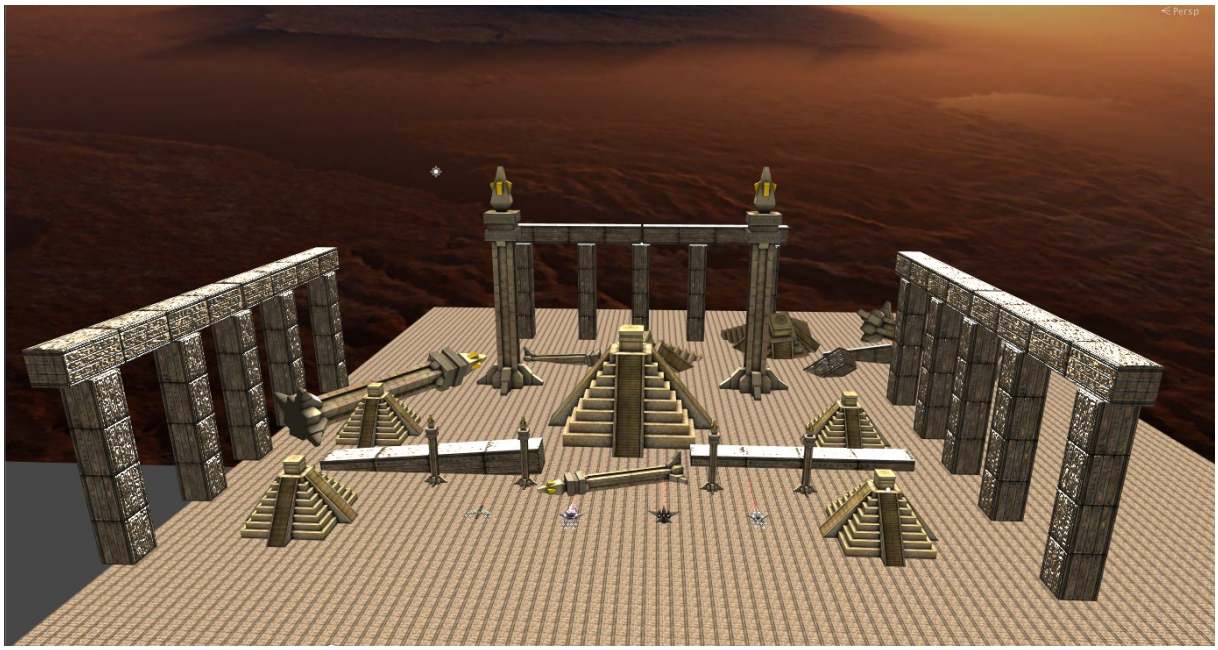
- Teksturo in barvo besedila za gumb v normalnem, oziroma mirujočem stanju.
- Teksturo in barvo besedila ob kliku na gumb.
- Font za besedila in njegovo velikost, poravnanost in še mnogo drugih parametrov (Slika 18).
- Za večino gumbov imam tudi določeno teksturo, ki predstavlja besedilo na gumbu. Ta tekstura se v mojih primerih uporablja namesto samega besedila, ki je določen s fontom (Slika 18).

Za izdelovo textur sem uporabil spletni generator [17], po potrebi pa sem jih še dodelal s programom Paint.NET.



Slika 18: GUI Style, oziroma vizualni stil elementa gumb za operacijski sistem Android. Nekatere parametre pri razvoju mobilnih aplikacij ni potrebno pokriti, ker se ne morejo zgoditi, na primer stanje lebdenja nad gumbom, ki ga označuje parameter *Hoover*. Slika prikazuje nastavitve in pripadajoče teksture za gumb Start. V tem primeru ni uporabljen font ampak tekstura z besedilom "Start". Izgled gumba je prikazan na sliki 20, postavitev gumba s kodo pa s Primerom 3.

4.1 Prvi igralni prostor



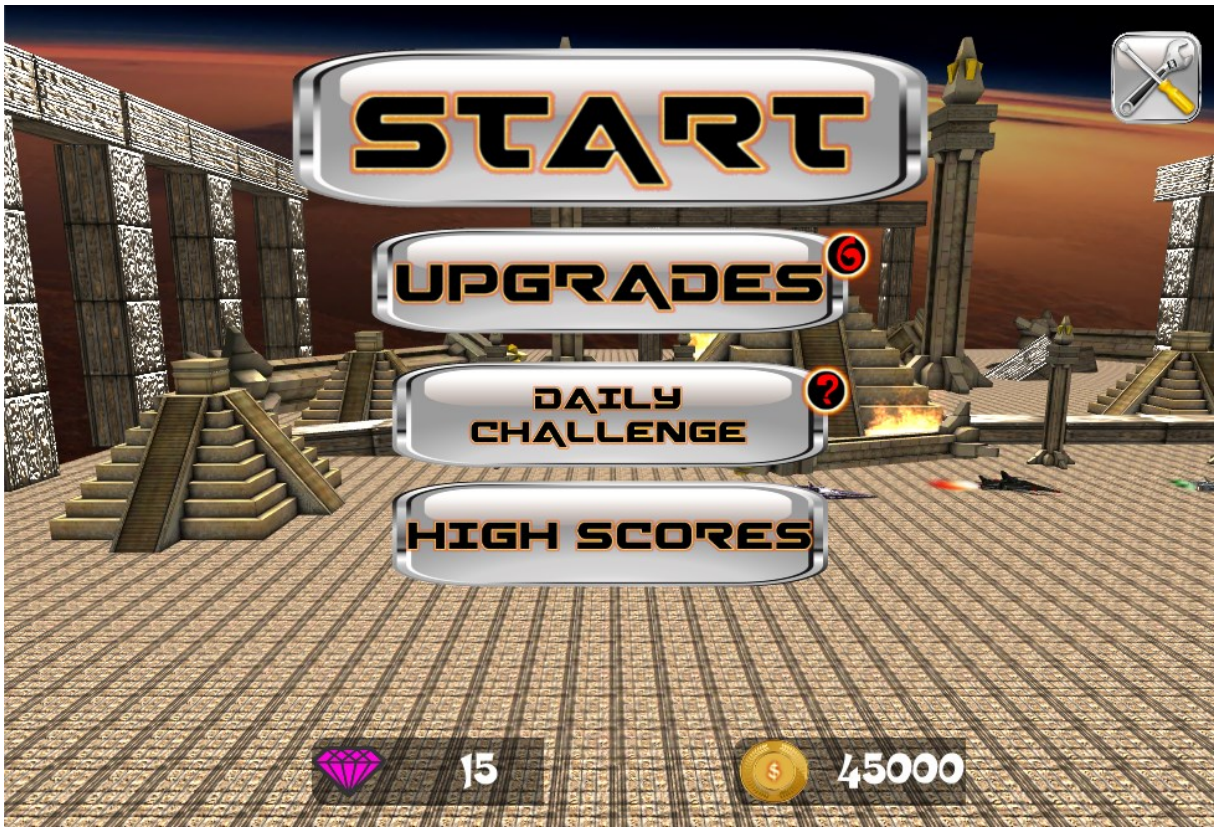
Slika 19: Pogled na prvi igralni prostor iz urejevalnika Unity.

Prvi igralni prostor (Slika 19) v celoti obvladuje skripta *ManageFirstWorld.cs*, ki je pritrjena na glavno kamero. Skripta skrbi za premikanje kamere po prostoru, prikazovanje GUI elementov, branje in shranjevanje vrednosti parametrov (število pobranih zlatnikov, najboljši rezultati, stopnje nadgradenj, ...), upravlja tudi postavitev nekaterih objektov v igralno polje.

Postavitev elementov GUI se lahko nahaja v več različnih stanjih, odvisno od tega, kje se igralec trenutno nahaja. Večino dogajanja v tem igralnem polju se torej opravlja znotraj funkcije *OnGui()*. Poleg sprememb v postavitvah elementov GUI, se v nekaterih primerih spremeni tudi pozicija kamere.

```
private enum GuiModes
{
    MainMenu,          // Slika 20
    Settings,          // Slika 23
    BestScores,        // Slika 25
    ConfirmPurchase,
    PlaneSelection,    // Slika 21
    Upgrades,          // Slika 22
    ModeSelection,
    MissionSelection,
    ShowDailyMission // Slika 26
}
```

Primer 2: različni načini postavitev GUI elementov v prvem igralnem prostoru. Vsaka postavitev omogoča igralcu določen nabor možnosti.



Slika 20: Glavni meni prvega igralnega prostora.

```
public Texture btnStart;
public Texture btnBestScores;
public Texture btnDailyMission;
public Texture btnSettings;
public Texture btnBestScores;

public GUIStyle CoinGUIStyle;
public GUIStyle DiamondGUIStyle;
public GUIStyle ButtonStyle;

privateint btnWidth = Screen.width / 3;
privateint btnHeigth = Screen.height/12;

void OnGUI()
{
    if (tmpLevel == SettingsLevel.Main)
    {
        if (GUI.Button(new Rect(Screen.width / 2 - btnWidth * 0.8f,
                                10,
                                btnWidth * 1.6f,
                                btnHeigth * 1.2f),
                       btnStart,
                       ButtonStyle))
        {
            audio.PlayOneShot(ButtonClick);
            MoveButtons(SettingsLevel.PlaneSelection);
            StartCoroutine(GoToPlaneSelectionPoint());
        }


        if (GUI.Button(new Rect(Screen.width / 2 - btnWidth * 0.6f,
                                btnHeigth + 25,
                                btnWidth * 1.2f,
                                btnHeigth * 0.8f),
                       btnUpgrades,
                       ButtonStyle))
        {
            audio.PlayOneShot(ButtonClick);
            MoveButtons(SettingsLevel.Upgrades);
            StartCoroutine(GoToUpgrades());
        }

        if (GUI.Button(new Rect(Screen.width / 2 - btnWidth * 0.55f,
                                btnHeigth * 2 + 25,
                                btnWidth * 1.1f,
                                btnHeigth * 0.8f),
                       btnDailyMission,
                ButtonStyle))
        {
                audio.PlayOneShot(ButtonClick);
                MoveButtons(SettingsLevel.ShowDailyMission);
        }
```

```
        if (GUI.Button(new Rect(Screen.width / 2 - btnWidth * 0.55f,
                                btnHeigth * 3 + 25,
                                btnWidth * 1.1f,
                                btnHeigth * 0.8f),
                        btnBestScores,
                        ButtonStyle))
        {
            audio.PlayOneShot(ButtonClick);
            MoveButtons(SettingsLevel.BestScores);
        }

        if(GUI.Button(new Rect(Screen.width - btnHeigth * 0.7f,
                               btnHeigth * 0.7f + 25,
                               btnHeigth * 0.7f,
                               btnHeigth * 0.7f),
                      btnSettings,
                      ButtonStyle))
        {
            audio.PlayOneShot(ButtonClick);
            MoveButtons(SettingsLevel.Settings);
        }

        //Prikaz št. zlatnikov.
        GUI.Label(new Rect(Screen.width/2 + btnHeigth - 10,
                           Screen.height - btnHeigth/2 - 20,
                           btnHeigth * 1.7f,
                           btnHeigth / 2),
                  NumberOfCoins.ToString(),
                  CoinGUIStyle);

        //Prikaz št. dijamantov.
        GUI.Label(new Rect(Screen.width/2 - 2 * btnHeigth - 20,
                           Screen.height - btnHeigth/2 - 20,
                           btnHeigth * 1.7f,
                           btnHeigth / 2),
                  NumberOfDiamonds.ToString(),
                  DiamondGUIStyle);

                .
                .
                .
        }
        .
        .
        .
}//end OnGUI()
```

Primer 3: Del kode iz skripte ManageFirstWorld.cs, ki določa postavitev in prikaz GUI elementov za način MainMenu.

Prvi igralni prostor omogoča:

- **Izbor letala:** igralec ima na voljo štiri različna letala (Slika 36). Na začetku je odklenjeno le prvo, dodatna letala lahko odklene z zlatniki, ki jih pobere med igranjem.

Slika 21: Eno od štirih letal in pripadajoči GUI. Letalo na sliki je še zaklenjeno. V spodnjem levem kotu so vidni trenutni parametri letala. Ko je letalo odklenjeno, mu je možno nadgraditi število raket in število ščitov. Možno je tudi odkleniti laserski namerilec in nadgraditi število raketometov.

- **Nadgradnje nagrad:** S pobranimi zlatniki lahko igralec nadgrajuje nagrade (Slika 22). Na primer: nadgrajeni paket raket vsebuje več raket, nadgrajeni magnet deluje dlje časa,...

Slika 22: Na levi sliki so v paketu 3 rakete, na desni (nadgrajeni paket) pa jih je 6. Ti paketi se v drugem igralnem prostoru naključno generirajo in jih mora igralec pobrati, da jih lahko koristi.

- **Nastavitve:** Igralec si lahko izbere težavnost igre, kvaliteto grafike, občutljivost senzorja za gravitacijo in omogoča ali onemogoča zvoke (Slika 23).

Slika 23: Vse možne nastavitve v igri. Možno je vklopiti ali izklopiti zvok in glasbo v ozadju, izbiramo lahko med zahtevnostjo igranja (višja zahtevnost pomeni višjo hitrost letala). Nastavimo lahko kvaliteto grafike in občutljivost nagibanja naprave.

Spreminjanje kvalitete grafike

V urejevalniku Unity lahko natančno definiramo nivoje kvalitete grafike. Vsak nivo ima svoje nastavitve, ki lahko močno vplivajo na kvaliteto grafike in s tem tudi na procesorsko zahtevnost igre. Za mobilne igre je zato priporočljivo, da se izogibamo "dragim" elementom grafike. V svoji igri imam tri nivoje kvalitete grafike: nizko kvaliteto, srednje ter visoko kvaliteto. Slika 24 prikazuje nastavitve za visoko kvaliteto, primer 4 pa ukaz s kodo C#, ki nastavi določeno kvaliteto.

Slika 24: Nastavitve grafike [16], ki jih uporabljam v igri za visok nivo grafike.

```
private void SetGraphic(int graphicLevel)
{
    if (graphicLevel == 0)
    {
        /* Z ukazom SetQualityLevel(int index,
           bool applyExpensiveChanges) določim nivo grafike,
            ki sem ga predhodno definiral
           v urejevalniku Unity (Slika 24). */
        QualitySettings.SetQualityLevel(0, true);
    }
    else if (graphicLevel == 1)
    {
        QualitySettings.SetQualityLevel(3, true);
    }
    else
    {
        QualitySettings.SetQualityLevel(4, true);
    }
}
```

Primer 4: Funkcija s katero spreminjam kvaliteto grafike.

- **Pregled najboljših rezultatov:** tukaj lahko igralec vidi svoje najboljše rezultate, ki jih je dosegel v preteklosti (Slika 25).

Slika 25: Prikaz najvišjega števila točk in najdaljše razdalje, ki jo je dosegel igralec v eni igri. Število doseženih točk je odvisno od najdaljše razdalje in števila opravljenih misij. Najvišje število točk se izračuna kot zmnožek največje dosežene razdalje in števila opravljenih misij (če je število opravljenih misij enako nič, se množenje ne izvrši). V zgornjem primeru torej ni opravljena še nobena misija. Poleg prikaza točk, je tukaj še prikaz najdaljše razdalje, največ pobranih zlatnikov v eni igri, ter prikaz števila vseh porabljenih zlatnikov.

- **Pregled dnevne misije:** Vsak dan se naključno generira nova misija (Slika 26), če jo igralec izpolni**,** si pridobi nagrado, ki je prav tako generirana naključno.

Slika 26: Prikaz dnevne misije. V zgornjem primeru je to naloga pobrati 6 magnetov, nagrada pa je 500 zlatnikov. Prikazan je tudi trenutni napredek, ki se sešteva preko več iger (ni potrebno, da jo igralec izpolni v samo eni igri, mora pa jo v roku 24 ur).

- **Izbor načina igranja:** Zatem, ko si igralec izbere letalo, si mora izbrati še način igranja. Na voljo ima dva načina:
    - Igranje v neskončnost, kjer poskuša doseči čim daljšo razdaljo, igra se zaključi le takrat, ko igralec "umre".
    - Izpolnjevanje misij, kjer mora igralec izpolnjevati določene naloge (na primer: pobrati določeno število raket, prevoziti neko razdaljo, ...). Z uspešnim izpolnjevanjem misij, si igralec povečuje večkratnik doseženih točk in sicer z vsako izpolnjeno misijo se večkratnik poveča za ena.

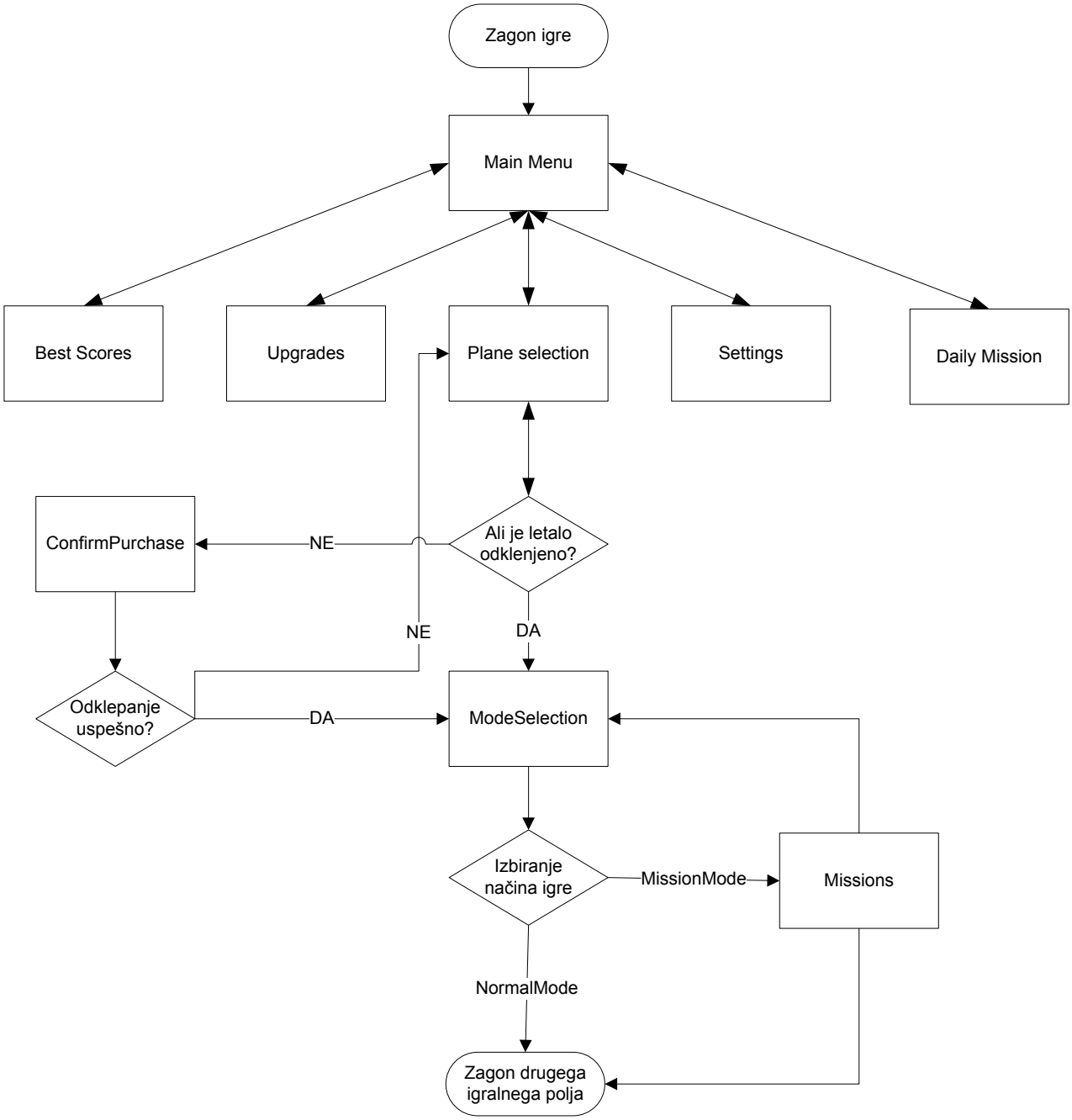
Ko si igralec izbere način igranja, se požene drugi igralni prostor.

```
public class ManageFirstWorld: MonoBehaviour
{
    //Dekleracija spremenljivk:
    public static int Plane1;
    public static int Plane2;
    public static int Plane3;
    public static int Plane4;

    private GuiModes tmpGUI;
    .
    .
    .

    void Start()
    {
        tmpGUI = MainMenu;

        /* Katera letala so že odklenjena in kako so nadrajena, katere
           misije so odklenjene, stopnje nadradenj nagrad,... */
        PreberiShranjeneVrednisti();

        /* Vporabi prebrane vrednosti za postavitev pravilnih objektov v
           igralni prostor in za pravilno prikazovanje vrednosti v
           elementih GUI. */
        UporabiPrebraneVrednosti();

    }

    void onGUI()/* Slika 27 predstavlja diagram
                   prehajanja vseh možnih postavitev GUI. */
    {
        If(tmpGUI == MainMenu)
        {
            Postavi_elemente_GUI_na_zahtevana_mesta();
            // Lovljenje pritiska na gumb.
            If(btnSettings)
            {
                Prestavi_kamero_na_položaj_za_Settings();
                tmpGUI = Settings;
```

```
            }
            .
            .
            .
    }
    else if(tmpGUI == Settings)
    {
        Postavi_elemente_GUI_na_zahtevana_mesta();
        // "Lovljenje" pritiska na gumb.
        If(btnMainMenu)
        {
            Prestavi_kamero_na_položaj_za_MainMenu();
            tmpGUI = MainMenu;
        }
        .
        .
        .
        }
        else if(tmpGUI == BestScores)
        .
        .
        .
    }

    private void PreberiShranjeneVrednisti()
    {
        /* Za shranjevanje in branje vrednosti spremenljivk v okolju Unity
        skrbijo metode v razredu PlayerPrefs[8]. */

        //Katera letala so že odklenjena?
        Plane2 = PlayerPrefs.GetInt("Plane2", 0);
        Plane3 = PlayerPrefs.GetInt("Plane3", 0);
        Plane4 = PlayerPrefs.GetInt("Plane4", 0);

        //Preberi stopnje nadgradenj letal.
        Plane2Rockets = PlayerPrefs.GetInt("Plane2Rockets", 0);
        Plane2Shields = PlayerPrefs.GetInt("Plane2Shields", 0);
        Plane2LaserSight = PlayerPrefs.GetInt("Plane2LaserSight", 0);

        Plane3Rockets = PlayerPrefs.GetInt("Plane3Rockets", 0);
        .
        .
        .

        //Preberi stopnje nadgradenj nagrad.
        Rockets = PlayerPrefs.GetInt("Rockets", 1);
        Magnet = PlayerPrefs.GetInt("Magnet", 1);
        MegaCoin = PlayerPrefs.GetInt("MegaCoin", 1);
        .
        .
        .
```

```
        //Preberi vrednosti nastavitev (iz Settings).
        GraphicQuality = PlayerPrefs.GetInt("GraphicQuality", 1);
        TiltSensitivity = PlayerPrefs.GetInt("TiltSensitivity", 50);
        Difficulty = PlayerPrefs.GetInt("Difficulty", 1);
        BackgroundMusic = PlayerPrefs.GetInt("BackgroundMusic", 1);
        Sounds = PlayerPrefs.GetInt("Sounds", 1);

        //Druge vrednosti.
        BestDistance = PlayerPrefs.GetInt("BestDistance", 1);
        CoinsSpent = PlayerPrefs.GetInt("CoinsSpent", 1);
        .
        .
        .
    }

    private void UporabiPrebraneVrednosti()
    {
        /* Preberi stopnje nadgradenj objektov in jih kreiraj v igralnem
           polju. */
        if (Rockets == 1)
        {
            // Rockets3, Slika 22.
            Instantiate(Rockets3,
                        new Vector3(24.15863f, 3.337884f, -21.69636f),
                        Quaternion.identity);
        }
         else if (Rockets == 2)
        {
             // Rockets6, Slika 22.
            Instantiate(Rockets6,
                         new Vector3(24.15863f, 3.337884f, -21.69636f),
                         Quaternion.identity);
        }
        .
        .
        .
    }
}
```

Primer 5: S simbolično kodo ponazorjena skripta *ManageFirstWorld.cs*.



Slika 27: Diagram prehajanja prikazov GUI v prvem igralnem prostoru.

4.2 Drugi igralni prostor

Med igranjem se letalo premika naprej po osi z, igralec pa ga lahko usmerja levo in desno (po osi x) z nagibanjem mobilne naprave. Igralno polje se gradi sproti, algoritem skrbi za to, da se ovire generirajo naključno. Ovire, ki jih letalo obvozi, se po določenem času odstranijo, da se sprosti pomnilniški prostor mobilne naprave.

Dogajanje v igralnem prostoru v največji meri določata skripti *BuildWorld.cs* in *Player.cs*. Skripta *BuildWorld.cs* skrbi za gradnjo igralnega polja, skripta *Player.cs* pa upravlja letalo.

4.2.1 Igralno polje

Za gradnjo in postavitev igralnega polja skrbi skripta *BuildWorld.cs*. Gradnja poteka dinamično glede na trenutno lokacijo letala, po principu: ko se letalo premakne za določeno število enot po osi z, pokliči funkcijo, ki "zgradi" naslednji blok igralnega polja. Gradnja igralnega polja poteka v treh korakih:

1. Postavi zidove, ki omejujejo gibanje letala.
2. Postavi ovire znotraj in zunaj zidov.
3. Postavi nagrade (zlatniki, diamanti, dodatne rakete, magnet...).

```
/* Funkcije tipa IEnumerator omogočajo izvajanje s časovnim zamikom, klic
teh funkcij mora potekati na sledeč način: StartCorutine(ImeFunkcije()).
[4]*/
IEnumerator Updater()
{
    while (true)
    {
        /* tmpDistance je spremenljivka, ki določa kdaj se bo zgradil
        naslednji blok igralnega polja. Gledamo za 200 enot naprej
        po osi z. */
        if (tmpDistance < Player.zPos)
        {
            // Klic funkcije, ki "postavlja" zidove.
            createWall();

            // Počakamo 0.1 sekunde, da ne preobremenimo procesorja.
            yield return new WaitForSeconds(0.1f);

            // Klic funkcije, ki "postavlja" ovire.
            StartCoroutine(createObjects());
            yield return new WaitForSeconds(0.1f);

            // Klic funkcije, ki "postavlja" nagrade v igralno polje.
            createRevards();

            // Povečamo vrednost spremenljivke tmpDistance.
            tmpDistance += 200;
```

```
            }

            // Počakamo 0.2 sekunde in gremo v ponovno preverjanje.
            yield return new WaitForSeconds(0.2f);
        }
    }
```

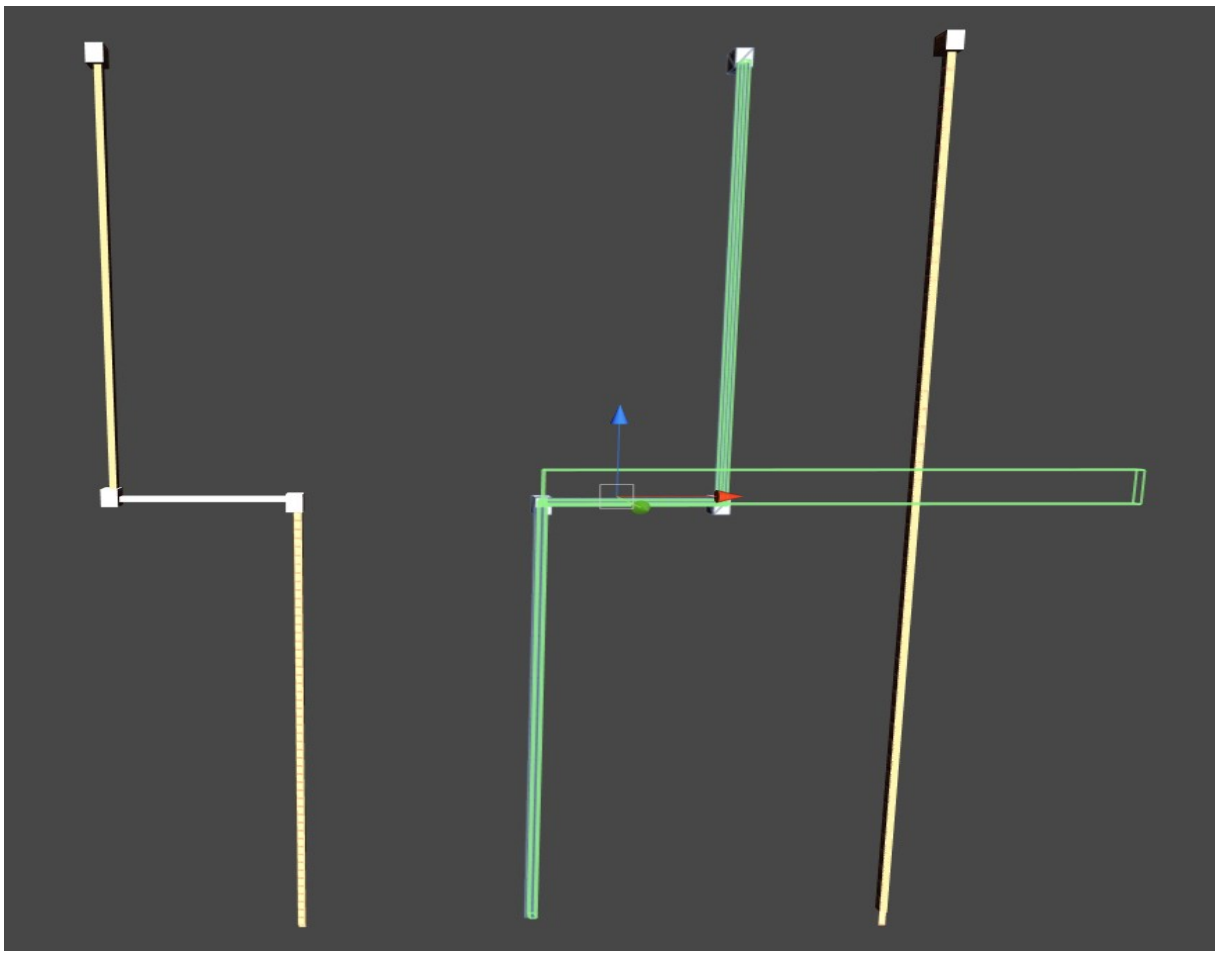
Primer 6: Funkcija *Updater()*, ki skrbi za gradnjo celotnega igralnega polja in je del skripte *BuildWorld()*.

4.2.2 Postavitev zidov

Za postavitev zidov v igralnem polju skrbi funkcija *createWall()*. Funkcija postavlja zid ločeno na levi in desni strani. Pri tem ima na voljo po tri tipe zidov za vsako stran (Slika 28):

1. Zid, ki igralni prostor razširi.
2. Zid, ki igralni prostor zoži.
3. Zid, ki ne spreminja širine igralnega prostora.

Dolžina vseh treh tipov zidov je 100 enot, zaradi poenostavitve algoritma. Funkcija sestavlja zid z naključnim izborom (z določenimi omejitvami) osnovnih tipov zidov.



Slika 28: Trije osnovni tipi zidov iz katerih algoritem sestavlja celoten "neskončni" zid. Na sredini je viden poleg zidu tudi pripadajoči trkalnik, ki ob trku določi meje koordinat x, do katerih se letalo še lahko giblje.

```
// Spremenljivke:
public GameObject lWall;
public GameObject lWallWide;
public GameObject lWallNarrow;

private enum LastLwall
{
    Normal,
    Wide,
    Narrow
}

/* Za postavitev naslednega dela zidu moramo vedeti, kateri del je bil
postavljen predhodno. */
private LastLwall lastLwall;

// Meja na levi strani, do koder se še lahko postavljajo ovire.
private int maxLeftDistance;

// Funkcija, ki "postavlja" zid.
```

```
private void createWall()
{
    // Vrednost leftRandom je lahko 1 ali pa 2.
    int leftRandom = UnityEngine.Random.Range(1, 3);
    int rightRandom = UnityEngine.Random.Range(1, 3);

    #region Left Wall
    if (leftRandom == 1)
    {
        if (lastLwall == LastLwall.Normal)
        {
            if (!isLeftWide)
            {
                Instantiate(lWall,
                            new Vector3(-1.00f,
                                        -0.8f,
                                        tmpDistance + 300),
                            Quaternion.identity);
                lastLwall = LastLwall.Normal;
                maxLeftDistance = 0;
            }
            else
            {
                Instantiate(lWall,
                            new Vector3(-16.00f,
                                        -0.8f,
                                        tmpDistance + 300),
                            Quaternion.identity);
                lastLwall = LastLwall.Normal;
                maxLeftDistance = -15;
            }
        }
        else if (lastLwall == LastLwall.Wide)
        {
            leftRandom = UnityEngine.Random.Range(1, 3);
            if (leftRandom == 1)
            {
                Instantiate(lWall,
                            new Vector3(-16.00f,
                                        -0.8f,
                                        tmpDistance + 300),
                            Quaternion.identity);

                lastLwall = LastLwall.Normal;
                isLeftWide = true;
                maxLeftDistance = -15;
            }
            else
            {
                Instantiate(lWallNarrow,
                            new Vector3(-10.0f,
                                        -0.8f,
```

```
                                        tmpDistance + 300),
                            Quaternion.identity);

                lastLwall = LastLwall.Narrow;
                maxLeftDistance = 0;
            }
        }
        else if (lastLwall == LastLwall.Narrow)
        {
             Instantiate(lWall,
                        new Vector3(-1.00f,
                                    -0.8f,
                                    tmpDistance + 300),
                        Quaternion.identity);

            lastLwall = LastLwall.Normal;
            maxLeftDistance = 0;
        }
    }
    else
    {
        if (lastLwall == LastLwall.Normal)
        {
            if (!isLeftWide)
            {
                Instantiate(lWallWide,
                            new Vector3(-2.2f,
                                        -1.66f,
                                        tmpDistance + 300),
                            Quaternion.identity);

                isLeftWide = true;
                lastLwall = LastLwall.Wide;
                maxLeftDistance = -15;
            }
            else
            {
                Instantiate(lWallNarrow,
                            new Vector3(-10.0f,
                                        -0.8f,
                                        tmpDistance + 300),
                            Quaternion.identity);

                lastLwall = LastLwall.Narrow;
                isLeftWide = false;
                maxLeftDistance = 0;
            }
        }
        else if (lastLwall == LastLwall.Wide)
        {
            leftRandom = UnityEngine.Random.Range(1, 3);
            if (leftRandom == 1)
```

```
                {
                    Instantiate(lWall, new Vector3(-16.00f, -0.8f,
                       tmpDistance + 300), Quaternion.identity);
                    lastLwall = LastLwall.Normal;
                    isLeftWide = true;
                    maxLeftDistance = -15;
                }
                else
                {
                    Instantiate(lWallNarrow,
                                new Vector3(-10.0f,
                                            -0.8f,
                                            tmpDistance + 300),
                                Quaternion.identity);

                    lastLwall = LastLwall.Narrow;
                    maxLeftDistance = 0;
                }
            }
            else if (lastLwall == LastLwall.Narrow)
            {
                Instantiate(lWall,
                            new Vector3(-1.0f,
                                        -0.8f,
                                        tmpDistance + 300),
                            Quaternion.identity);

                lastLwall = LastLwall.Normal;
                maxLeftDistance = 0;
            }
        }
        #endregion
.
.
.
}
```

Primer 7: Del algoritma, ki sestavlja levo stran zidov.

4.2.3 Postavitev ovir

V verziji 1.1 je v igri 23 različnih ovir, algoritem jih izbira naključno in postavlja v igralno polje na določena mesta (ta mesta so določena glede na trenutno pozicijo letala in trenutno pozicijo zidov). Ovire se v osnovi delijo na statične, ki se ne premikajo, in dinamične, na katere lahko deluje gravitacija ali pa jih premikam preko kode. Primer statične ovire je piramida (Slika 30), primer dinamične pa komet (Slika 31), ki je naključno generiran nekje "v zraku" in se premika proti površini dokler ne doseže določene vrednosti koordinate y (Primer 9).

Statične ovire: so ovire, ki mirujejo v igralnem polju. Na njih ne deluje fizikalni pogon, niti se ne premikajo zaradi kode. Z igralčevega vidika so to: piramide, stebri, stene...

Dinamične ovire: so ovire, na katere deluje fizikalni pogon ali pa jih premika koda. Z igralčevega vidika so to: stebri, ki se podirajo, kometi, krogle in vse ostalo, kar se v igri premika.

Sestavljene ovire (Slika 29): so skupek najmanj dveh ovir, ni pomembno ali sta statični ali dinamični ali kombinacija obojega.

Funkcija, ki skrbi za postavljanje ovir v igralno polje se nahaja v skripti *BuildWorld.cs* in se imenuje *createObstacles()*.

```
tabela_objektovOvirZunajZidov;
tabela_objektovOvirZnotrajZidov;

#region ZunajZidov
        naključno izberi dva objekta iz tabele
        tabela_objektovOvirZunajZidov;
        postavi prvi objekt na levo stran zidov; /* Pozicija se določi
        glede na trenutno lokacijo letala. */
        postavi drugi objekt na desno stran zidov;
#endregion

#region ZnotrajZidov
        naključno izberi objekt iz tabele tabela_objektovOvirZnotrajZidov;
        postavi izbrani objekt znotraj zidov;
#endregion
```

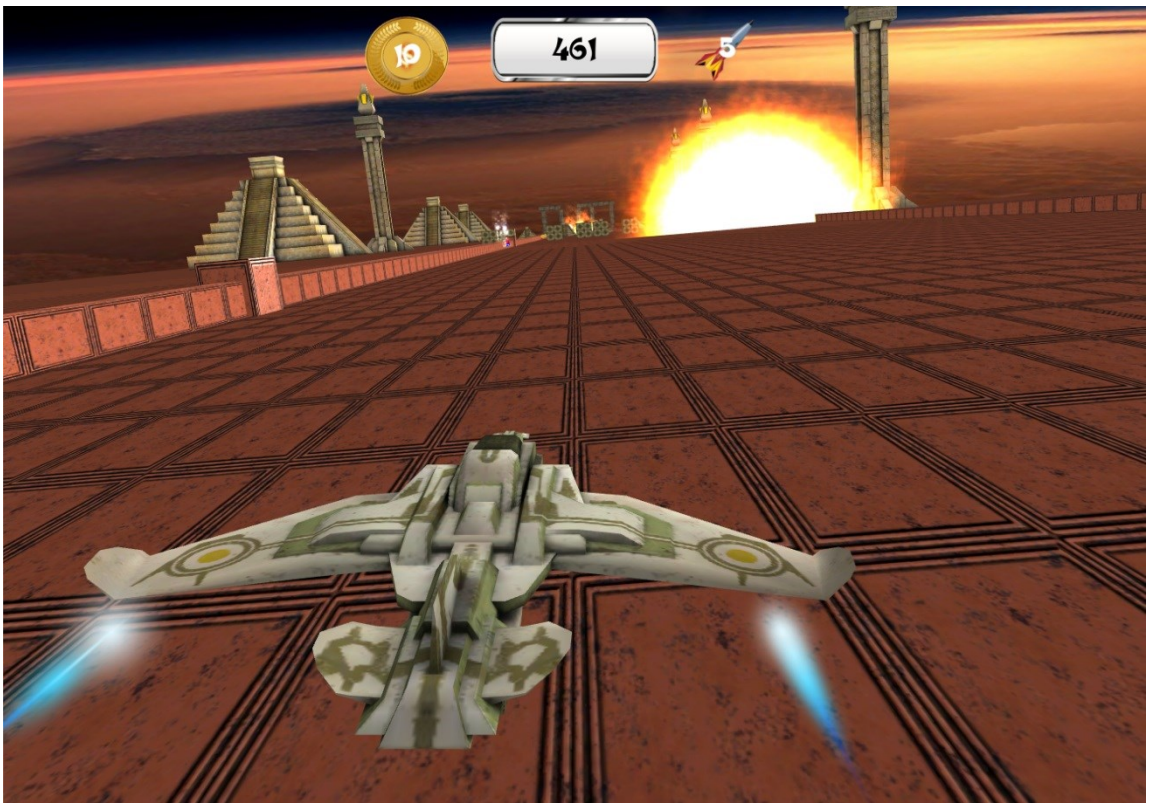
Primer 8: Postopek postavljanja ovir v igralni prostor.

Slika 29: Primer sestavljene ovire. Na levem delu slike je posnetek ovire med igranjem, na desni pa predogled v orodju Unity. Ovira je kombinacija dinamičnih in statičnih objektov.

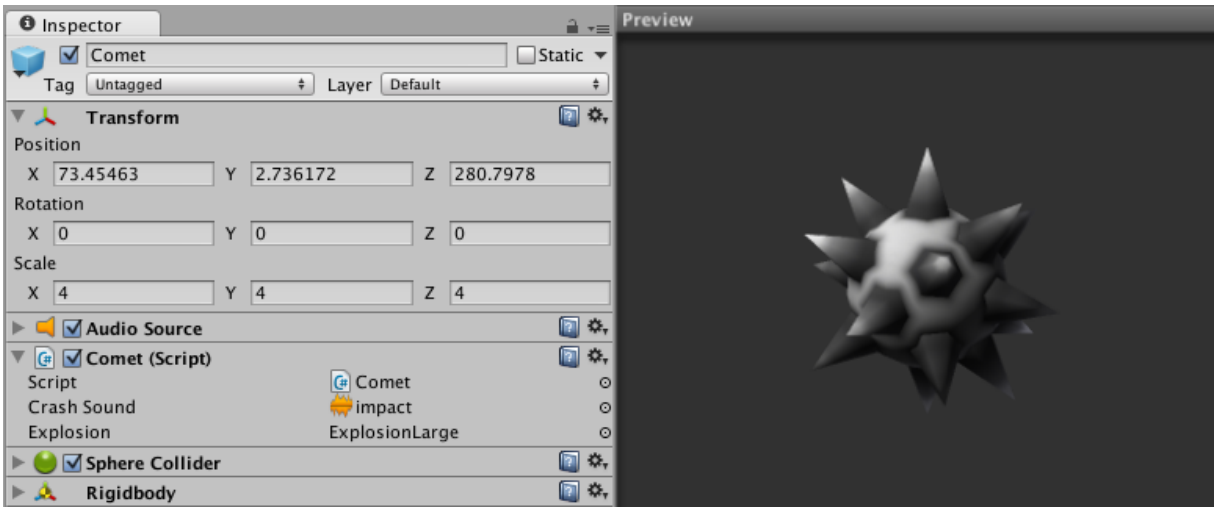
Ovira s slike 29 je sestavljena iz:

- 15 dinamičnih stebrov, na katere deluje gravitacija in se zato lahko podrejo, če jih kaj zadane.
- Statična piramida na sredini.
- Sistem za simulacijo delcev, ki simulira ogenj (viden na levem delu slike, znotraj stebrov).

Vsi stebri in piramida imajo dodane trkalnike, zato se jim mora igralec izogniti. Pri tej sestavljeni oviri ni dodane nobene kode, vso potrebno funkcionalnost vodi pogon Unity.



Slika 30: Padec kometa na površino. Komet spada med dinamične ovire in je v celoti voden s kodo.



Slika 31: Sestava kometa: poleg ostalih komponent je dodana tudi skripta Comet.cs, ki določa celotno delovanje objekta komet.

```
using UnityEngine;
public class Comet : MonoBehaviour
{
        public AudioClip CrashSound;
        public GameObject Explosion;

        private Transform myTransform;
        private int xRot = 1;
        private int yRot = 1;
        private int zRot = 1;
        private int myScale = 4;
        private int speed = 100;
        private Vector3 endPoint;

        // Inicializacija spremenljivk.
        void Start ()
        {
                // Naključno izberemo hitrost rotacije okoli osi x, y in z.
                xRot = UnityEngine.Random.Range(3, 9);
                yRot = UnityEngine.Random.Range(3, 9);
                zRot = UnityEngine.Random.Range(3, 9);

                // Naključno izberemo velikost kometa.
                myScale = UnityEngine.Random.Range(4, 9);

                // Naključno izberemo hitrost premikanja kometa.
                speed = UnityEngine.Random.Range(160, 240);

                // Določimo točko, kamor naj komet pade.
                endPoint = new Vector3(UnityEngine.Random.Range(-20, 40),
                                       -2,
                                       Player.zPos +
                                       UnityEngine.Random.Range(50, 200));

                /* Komponento transform prepišemo v lokalno spremenljivko,
                zaradi optimizacije. */
                myTransform = transform;

                // Določimo velikost kometa.
                myTransform.localScale = new Vector3(myScale,
                                                     myScale,
                                                     myScale);

                /* Pokličemo funkcijo StartTravel(), ki vodi gibanje kometa
                od začetne, do končne točke. */
                StartCoroutine(StartTravel());
        }

        IEnumerator StartTravel()
        {
                /* Premikaj in rotiraj komet, dokler je y kooordinata
                več kot -1. Ko doseže koordinata y vrednost -1 ali manj,
```

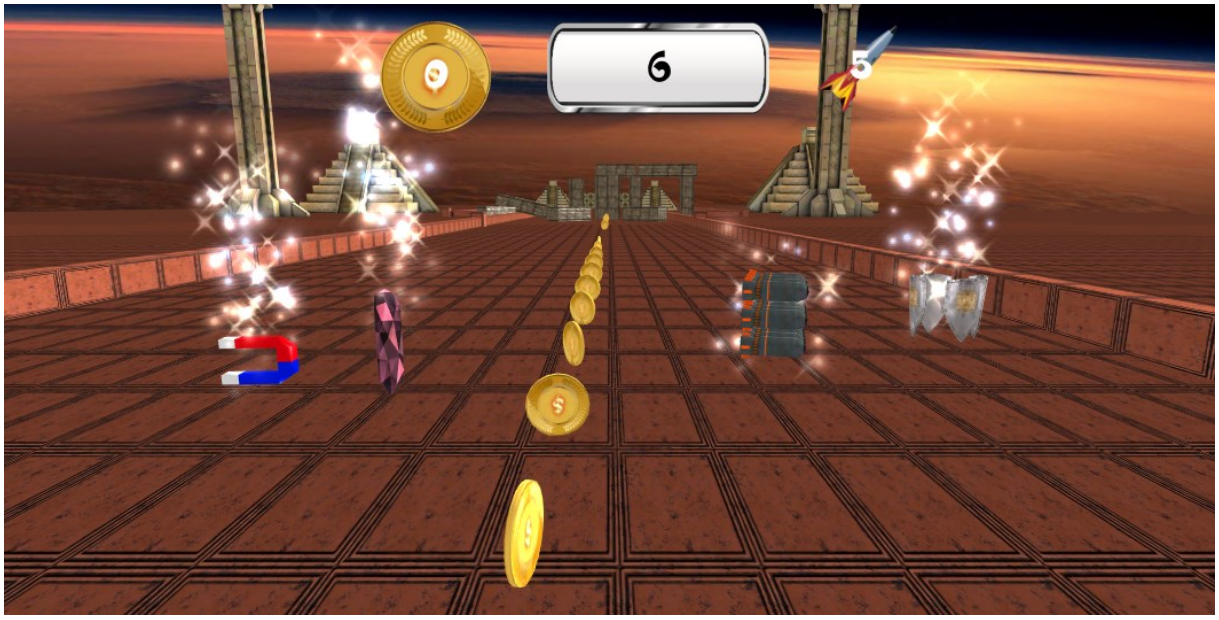
```
                je komet na površini. */
                while(myTransform.position.y > -1)
                {
                        myTransform.position =
                                Vector3.MoveTowards(myTransform.position,
                                                endPoint,
                                                speed * Time.deltaTime);

                        myTransform.Rotate(new Vector3(xRot, yRot, zRot));
                        yield return null;
                }

                // Komet je "padel" na površino, sedaj sproži eksplozijo.
                Instantiate(Explosion,
                            new Vector3(myTransform.position.x,
                                        myTransform.position.y,
                                        myTransform.position.z - 5),
                            Quaternion.identity);

                // Če so zvoki omogočeni, sproži tudi zvok.
                if(soundsEnabled)
                    audio.PlayOneShot(CrashSound);

                /* Poišči skripto Player.cs in pokliči funkcijo
                ShakePlayer(), s tem simuliramo potres. Skripta Player.cs
                mora nujno obstajati v igralnem prostoru, da jo lahko
                kličemo, drugače se program sesuje. */
                Player player = (Player)FindObjectOfType(typeof(Player));
                player.ShakePlayer();

        }

        /* Funkcija OnTriggerEnter(Collider other) je rezervirana funkcija
        MonoBehaviour razreda. Pokliče se, kadar en trkalnik zadane
        drugega, ki je sprožilec (ang. trigger). */
        void OnTriggerEnter(Collider other)
        {
                /* Če ima zadeti objekt označbo "piramid_small" ali
                "pillar", sproži eksplozijo in zvok, ter uniči objekt. */
                if(other.tag == "piramid_small" || other.tag == "pillar")
                {
                        Instantiate(Explosion, other.transform.position,
                        Quaternion.identity);
                        if(soundsEnabled)
                                audio.PlayOneShot(CrashSound);
                        Destroy(other.gameObject);
                }
        }
}
```

Primer 9: Skripta, ki upravlja komet.

4.2.4 Postavitev nagrad

Za postavitev nagrad v igralno polje skrbi funkcija *createRewards()* in je del skripte *BuildWorld.cs*. Nagrade izbira naključno, ter jih postavlja v igralno polje na določene lokacije, ki so odvisne od trenutnega položaja letala. V igri je 8 različnih tipov nagrad (Slika 32):

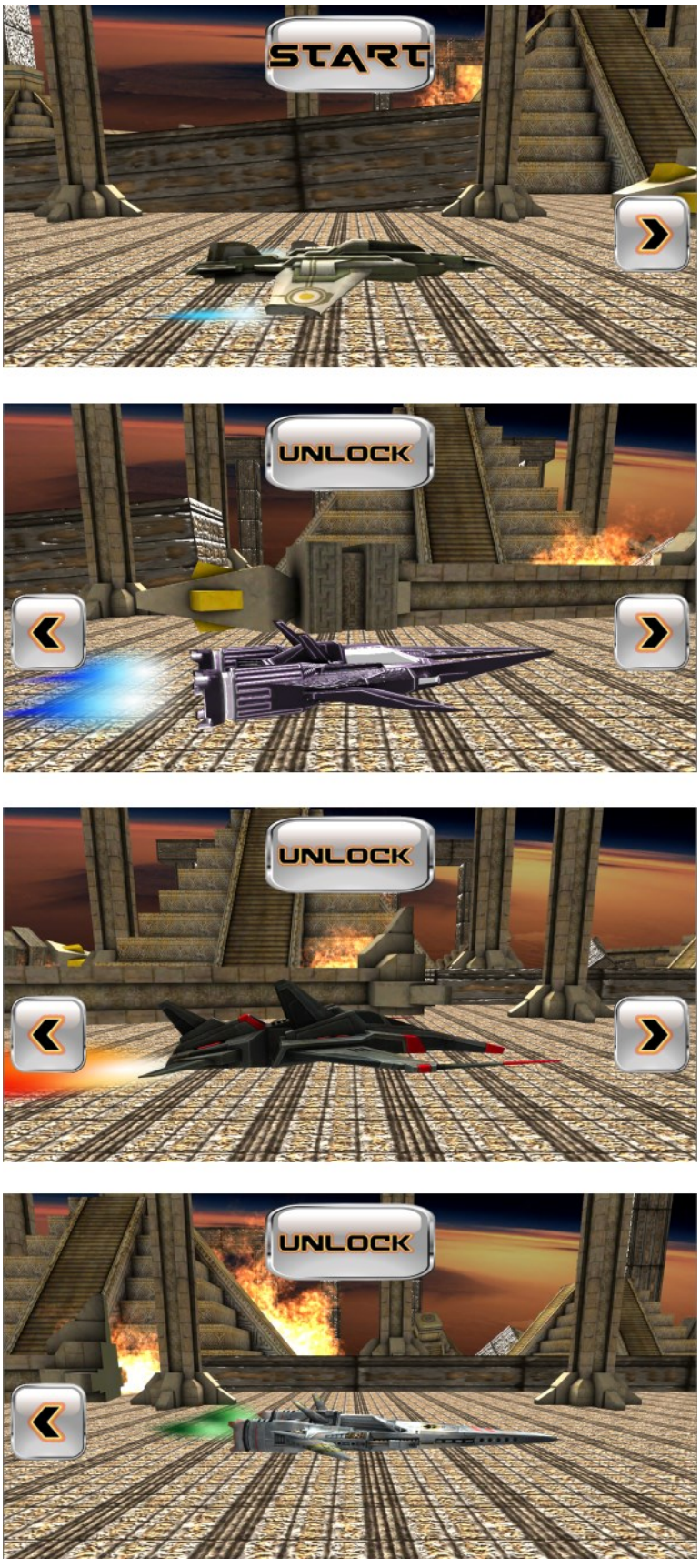
- Zlatniki: omogočajo igralcu, da nadgrajuje ali pa kupi boljše letalo.
- Magnet: za določen čas samodejno pobira zlatnike.
- Rakete: igralec lahko z njimi "razbije" nekatere ovire in zaradi tega lažje preživi.
- Nesmrtnost: igralec lahko potuje skozi ovire brez poškodb.
- Ščit: obnovi ščit letala. Kolikor ščitov ima igralec, tolikokrat lahko trči v ovire, preden je igre konec.
- Mega zlatnik: ima vrednost od 25 do 150 navadnih zlatnikov.
- Diamant: omogoča aktivacijo posebnih lastnosti letala (trenutno je to le nesmrtnost).



Slika 32: Nagrade v igralnem polju.

4.2.5 Letalo

V igri so na voljo štiri različna letala (Slika 33). Igralec ima na začetku odklenjeno le prvo. Med igranjem pobira zlatnike, s temi zlatniki pa lahko odklene še dodatna tri boljša letala (imajo več raket in boljši ščit). Vsakemu letalu je možno nadgrajevati ščit in število raket.



Slika 33: Letala v igri Temple Attack.

Sestava letal

Sestava letala se razlikuje glede na to, ali se letalo nahaja v prvem ali v drugem igralnem prostoru. V prvem igralnem prostoru se letalo le vrti na mestu, v drugem pa mu je omogočeno dosti bolj kompleksno gibanje in interakcija z okolico. Vrtenje na mestu omogoča preprosta skripta *Rotate.cs* (Primer 10), gibanje in interakcijo v drugem igralnem prostoru pa skripta *Player.cs*. Razlike so podrobneje opisane spodaj.

Sestava letala v prvem igralnem prostoru:

- 3D model s pripadajočimi teksturami.
- Izrisovalec 3D modela.
- Skripta *Rotate.cs*.
- Simulator delcev, ki simulira izpušni ogenj letala.

```
using UnityEngine;

public class Rotate : MonoBehaviour
{
    private Transform myTransform;

    void Start()
    {
        myTransform = transform;
    }

    void Update()
    {
        myTransform.Rotate(new Vector3(0, 25, 0) * Time.deltaTime);
    }
}
```

Primer 10: Skripta *Rotate.cs*, ki je pritrjena na letala v prvem igralnem prostoru. Njena naloga je rotirati letala preko osi y.

Sestava letala v drugem igralnem prostoru:

- 3D model s pripadajočimi teksturami.
- Izrisovalec 3D modela.
- Togo telo (ang. RigidBody).
- Trkalnik za zaznavanje trkov z drugimi objekti.
- Vir zvoka (ang. Audio Source).
- Glavna kamera.
- Skripta Player.cs
- Magnet, ki je na začetku neaktiviran oziroma onemogočen. Aktivira se za določen čas, kadar igralec v igralnem polju pobere magnet.

- Simulator delcev, ki simulira izpušni ogenj letala.

3D model s pripadajočimi teksturami (Slika 34)

Unity podpira naslednje formate 3D modelov: .FBX, .dae (Collada), .3DS, .dxf in .obj. 3D modeli, ki jih uporabljam v igri so različnih formatov (v istem projektu lahko imamo različne formate 3D modelov). 3D modelov letal nisem izdelal sam, ampak sem jih pobral z različnih spletnih portalov, ki ponujajo veliko brezplačnih in tudi plačljivih 3D modelov. Za projekt sem uporabljal le brezplačne. Največji portali z 3D modeli so: http://www.turbosquid.com/, http://archive3d.net/, http://www.3dxtras.com/, in še mnogo drugih.

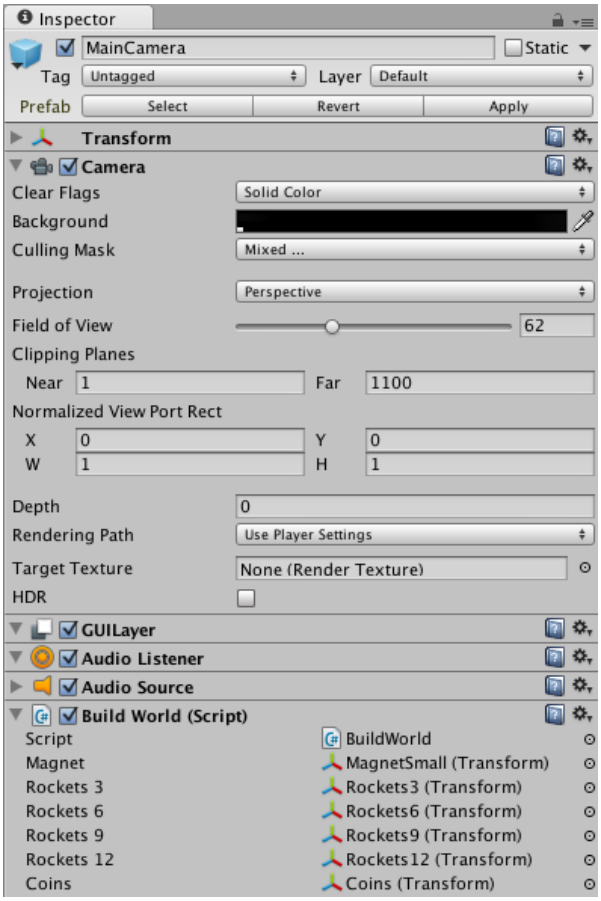
Slika 34: Primer 3D modela letala s pripadajočo teksturo. 3D model je formata 3DS, tekstura pa jpg.

Glavna kamera

Glavna kamera je na letalo pritrjena kot potomec. To pomeni, da se kamera samodejno premika in rotira enako, kot letalo.

Nanjo so pritrjene komponente (Slika 35):

- Transform.
- Kamera s pripadajočimi nastavitvami.
- GUILayer, ki omogoča prikaz oziroma izris komponent GUI na zaslon.
- Audio Listener omogoča sprejemanje zvokov.
- Audio Source: se uporablja za igranje glasbe iz ozadja.
- Skripta BuildWorld.cs.



Slika 35: Nastavitve glavne kamere drugega igralnega prostora.

Skripta Player.cs

Skripta vodi vso dogajanje, kar se zadeva letala. Upravlja premikanje letala skozi igralni prostor, ureja dogajanje ob trkih v posamezne elemente, prikazuje podatke (npr. koliko zlatnikov je igralec pobral v trenutni igri) na zaslonu. Ista skripta Player.cs je pritrjena na vsa 4 letala.

Vsako od letal se vedno nahaja v enem izmed štirih različnih stanj (Primer 11). Od teh stanj je odvisno "obnašanje" letala in prikaz GUI, ter delovanje igralnega polja. Ko je letalo na primer v *Paused* stanju, morajo mirovati tudi vsi ostali objekti v igralnem prostoru, to dosežem z ukazom *Time.timeScale = 0*.

```
enum PlayingMode
{
    PrePlay, //Letalo miruje, Time.timeScale = 0;
    Alive,   //Letalo se premika, Time.timeScale = 1;
    Dead,    //Letalo miruje, Time.timeScale = 0;
    Paused   //Letalo miruje, Time.timeScale = 0;
}
```

Primer 11: Vsa možna stanja v katerih se lahko letalo nahaja**.**

Premikanje letala

Premikanje letala se upravlja v funkciji Update(), letalo se z igralčevega vidika samodejno premika naprej po osi z, igralec pa ga z nagibanjem mobilne naprave usmerja levo in desno po osi x. Gibanje letala levo in desno navidezno omejuje zid, v resnici pa je gibanje omejeno le z vnaprej določenimi vrednostmi koordinate x, ki sovpadajo z x koordinatami zidu. Vrednost teh koordinat se določi s trkalnikom, ki je pritrjen na nekatere zidove (Slika 28).

```
void Update()
{
    if (playingMode == PlayingMode.Alive)
    {
        /* Pridobi željeno rotacijo letala ob nagibanju mobilne naprave. */
        Quaternion target = Quaternion.Euler(0f,
                                             0f,
                                             Input.acceleration.y * 90);

        myTransform.rotation = Quaternion.Lerp(myTransform.rotation,
                                               target,
                                               Time.deltaTime * 4f);

        /* Če se letalo nahaja znotraj meja zidov, ga premakni levo ali
        desno, odvisno od nagiba mobilne naprave. */
        if(lWallPosition < myTransform.position.x
            && rWallposition > myTransform.position.x)
        {
                myTransform.Translate(
                                new Vector3(
                                        Input.acceleration.y *
                                        ManageFirstWorld.TiltSensitivity,
                                        0.0f,
                                        PlayerSpeed) *
                                                Time.deltaTime,Space.World);

        }
        .
        .
        .
    }
}
```

Primer 12: Del kode iz skripte Player.cs, za premikanje letala.

Nadzor trkov

Vsakič, ko letalo trči v drug objekt, se sproži nek dogodek, odvisno od objekta v katerega letalo trči. Če letalo trči naprimer v kovanec, se mora zgoditi nekaj povsem drugega, kot pa če letalo trči v steber.

Funkcija, ki upravlja s trki v okolju Unity se imenuje *OnTriggerEnter()* in zahteva parameter tipa *Collider* (Primer 13). Funkcija se kliče vedno kadar se dva trkalnika dotakneta v vsaj eni točki 3D prostora. Da ugotovimo katerega objekta se je letalo dotaknilo, moramo dostopati do oznake drugega objekta. Poljubno oznako vsakemu objektu lahko določimo v urejevalniku Unity.

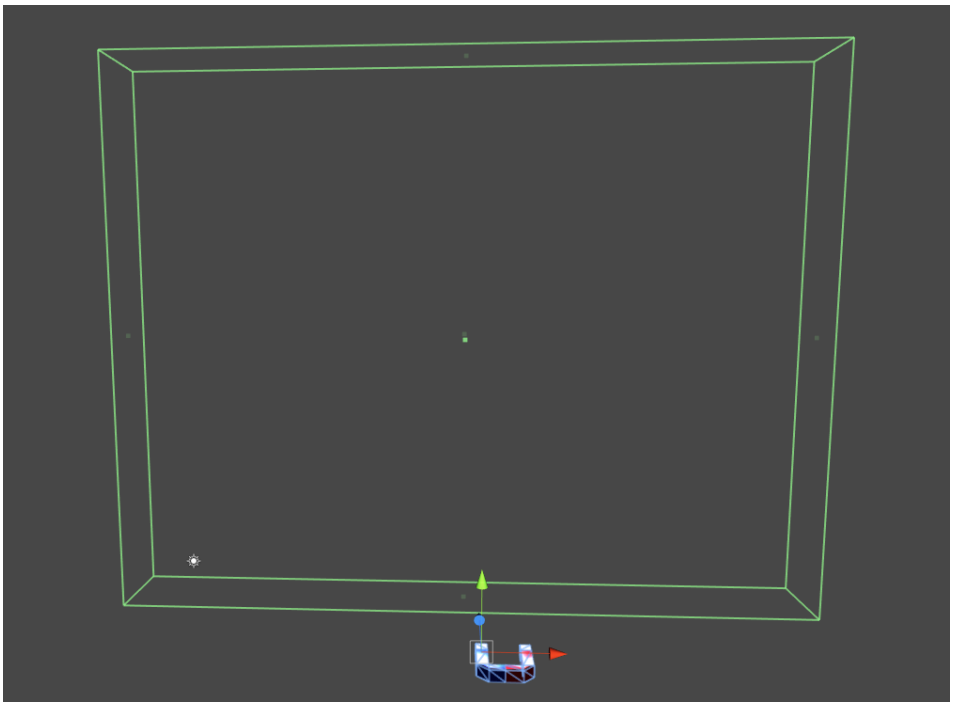
```
void OnTriggerEnter(Collider other)
{
        if (other.tag == "coin")
        {
            if (soundsEnabled)
                audio.PlayOneShot(coinPick, 10f);

            coinNumber++;
            Destroy(other.gameObject);
        }
        else if (other.tag == "magnet")
        {
            if (soundsEnabled)
                audio.PlayOneShot(magnetPick, 10f);

            StopCoroutine("MagnetMode");
            StartCoroutine("MagnetMode");
            Destroy(other.gameObject);
        }
        .
        .
        .

}  // End of OnTriggerEnter(Collider other).

IEnumerator MagnetMode()
{
        // Postavi magnet v aktivno stanje.
        Magnet.SetActive(true);

        /* Preberi stopnjo nadgradnje magneta. Od te stopnje je odvisen čas
        trajanja aktivnosti magneta */
        int level = PlayerPrefs.GetInt("MagnetLevel", 1);

        if (level == 1)
        {
            yield return new WaitForSeconds(10);
        }
        else if (level == 2)
```

```
        {
            yield return new WaitForSeconds(15);
        }
        else if (level == 3)
        {
            yield return new WaitForSeconds(20);
        }
        else if (level == 4)
        {
            yield return new WaitForSeconds(25);
        }

        Magnet.SetActive(false);
        yield return null;
}
```

Primer 13: Del kode iz skripte Player.cs, ki vodi dogodke ob trkih letala z drugimi objekti.

Magnet

Magnet (Slika 36) omogoča pobiranje zlatnikov na daljavo. Na začetku je onemogočen (ang. disabled). Igralec ga aktivira s tem, ko ga pobere v igralnem polju. Magnet ima svoj trkalnik in je pritrjen na letalo kot potomec.



Slika 36: Objekt magnet s pripadajočim trgalnikom. Ko se trkalnik dotakne zlatnika, se pokliče funkcija za pobiranje kovanca *GetCoin(Transform coinTransform)*.

```
void OnTriggerEnter(Collider other)
```

```
{
        if (other.tag == "coin")
        {
                StartCoroutine(GetCoin(other.transform));
        }
}

IEnumerator GetCoin(Transform coin)
{
    while (Vector3.Distance(coin.position, myTransform.position) > 3f)
    {
        coin.position = Vector3.Lerp(coin.position,
                                     new Vector3(myTransform.position.x,
                                                 myTransform.position.y,
                                                 myTransform.position.z+2),
                                     0.4f);

        yield return null;
    }

    if (soundsEnabled)
        audio.PlayOneShot(coinPick, 10f);

    coinNumber++;

    Destroy(coin.gameObject);
    yield return null;
}
```

Primer 14: Pobiranje kovanca z magnetom.

Izstrelitev raket

Igralec ima na začetku vsake igre neko določeno število raket, odvisno od nadgradenj. Z dotikom na zaslon lahko te rakete "izstreli" (Slika 5). Za samo izstrelitev skrbi skripta *Player.cs*, in sicer na način, da raketo postavi v igralno polje na določeno mesto pred letalo. Ko je raketa postavljena v igralno polje, prevzame nadzor nad njo skripta *Rocket.cs* (Primer 16), ki je pritrjena na raketo. Bistvo te skripte je pomikanje rakete naprej po osi z in detekcija trkov, ter reakcija ob trku.

```
private bool isRocketReady = true;
private int numberOfRockets = PlayerPrefs.GetInt("numberOfRockets", 1);
public AudioClip outOfRockets;

.
.
.
```

```
void Update()
{
    //Ob dotiku na zaslon, postavi raketo v igralno polje.
    if (Input.touchCount == 1)
    {

        if (isRocketReady)
            StartCoroutine(ShootRocket());
    }


private IEnumerator ShootRocket()
{
    if (numberOfRockets > 0)
    {
        /* myTransform določa pozicijo letala. Postavi raketo
        na položaj koordinat (x, y, z + 2), glede na trenutno
        lokacijo letala. */
        Instantiate(rocket,
                    new Vector3(myTransform.position.x,
                                myTransform.position.y,
                                myTransform.position.z + 2),
                    Quaternion.identity);

        numberOfRockets--;

        /* Za 0.5 sekunde onemogočim izstrelitev naslednje rakete. To
        predstavlja čas nalaganja nove rakete, drugače bi se vse
        rakete izstrelile v trenutku. */
        isRocketReady = false;
        yield return new WaitForSeconds(0.5f);
        isRocketReady = true;
    }
    else
    {
        if (soundsEnabled)
        {
            MainCamera.audio.PlayOneShot(outOfRockets, 10f);
            isRocketReady = false;
            yield return new WaitForSeconds(0.5f);
            isRocketReady = true;
        }
    }
}
```

Primer 15: Postavitev rakete v igralno polje (v skripti *Player.cs*).

```
using UnityEngine;
```

```
public class Rocket : MonoBehaviour
{
    private Transform myTransform;
    /* Različne oblike eksplozij. O
    public GameObject Explosion;
    public GameObject Explosion1;
    public GameObject Explosion2;
    public GameObject Explosion3;
    public GameObject ExplosionElectric;
    public AudioClip ExplosionClip;

    void Start ()
    {
        myTransform = transform;
    }

    void Update ()
    {
        /* Določi hitrost s katero se naj raketa premika. Hitrost rakete je
        odvisna od hitrosti letala (raketa je vedno za 10 hitrejša od
        letala). */
        float amtToMove = (Player.PlayerSpeed + 10f) * Time.deltaTime;

        // Premakni raketo naprej po osi z.
        myTransform.Translate(Vector3.forward * amtToMove);

        /* Ko je raketa po osi z za 250 oddaljena od letala, jo odstrani iz
        igralnega polja. Z odstranitvijo rakete, preneha delovati tudi
        skripta Rocket.cs. */
        if (myTransform.position.z > Player.zPos + 250f)
        {
            if(gameObject != null)
            Destroy(gameObject);
        }
    }

    void OnTriggerEnter(Collider other)
    {
        string myTag = other.gameObject.collider.tag;

        // Poglej v kateri objekt je trčila raketa in ustrezno ukrepaj.
        if (myTag == "pillar" || myTag == "piramid" ...)
        {
            Instantiate(Explosion,
                        myTransform.position,
                        Quaternion.identity);
        }
        else if (myTag == "piramid_small")
        {
            Destroy(other.gameObject);
            Instantiate(Explosion2, myTransform.position,
            Quaternion.identity);
```

```
        }
        .
        .
        .
        else
        {
             Destroy(other.gameObject);
            Instantiate(Explosion, myTransform.position,
            Quaternion.identity);

            /* Sproži zvok na mestu, kjer se je zgodil trk (bližje, kot se
            je zgodil trk, glasnejši bo zvok). */
           AudioSource.PlayClipAtPoint(ExplosionClip,myTransform.position);

          // Odstrani raketo.
          if(gameObject != null)
             Destroy(gameObject);
     }
}
```

Primer 16: Skripta *Rocket.cs*, ki upravlja z raketo.

4.2.6 Vizualni učinki

Poleg nastavitev kvalitete grafike, na vizualno podobo igre vpliva tudi jakost svetlobe in gostota ter barva megle (Slika 37). Za upravljanje megle imam napisani dve funkciji, ki naključno določata barvo in gostoto (Primer 17).

```
private IEnumerator HandleFog()
{
    RenderSettings.fogMode = FogMode.Exponential;
    RenderSettings.fog = true;
    RenderSettings.fogDensity = 0.0f;

    SetFogColor();

    /* Postopno večanje gostote megle. */
    while (RenderSettings.fogDensity < 0.05f)
    {
        RenderSettings.fogDensity += 0.0002f;
        yield return new WaitForSeconds(0.1f);
    }

    /* Megla naj bo prisotna naključno dolgo. */
    yield return new WaitForSeconds(Random.Range(5, 15));


    /* Postopno manjšanje gostote megle. */
```

```
    while (RenderSettings.fogDensity > 0.0f)
    {
        RenderSettings.fogDensity -= 0.0005f;
        yield return new WaitForSeconds(0.1f);
    }

    /* Odstrani meglo iz prostora. */
    RenderSettings.fog = false;
    yield return null;
}

private void SetFogColor()
{
    int fogColor = Random.Range(1, 6);

    if(fogColor == 1)
    {
        RenderSettings.fogColor = Color.white;
    }
    else if(fogColor == 2)
    {
        RenderSettings.fogColor = Color.gray;
    }
    else if(fogColor == 3)
    {
        RenderSettings.fogColor = Color.yellow;
    }
    else
        RenderSettings.fogColor = newColor(0.6f, 0.0f, 0.0f, 0.9f);
}
```

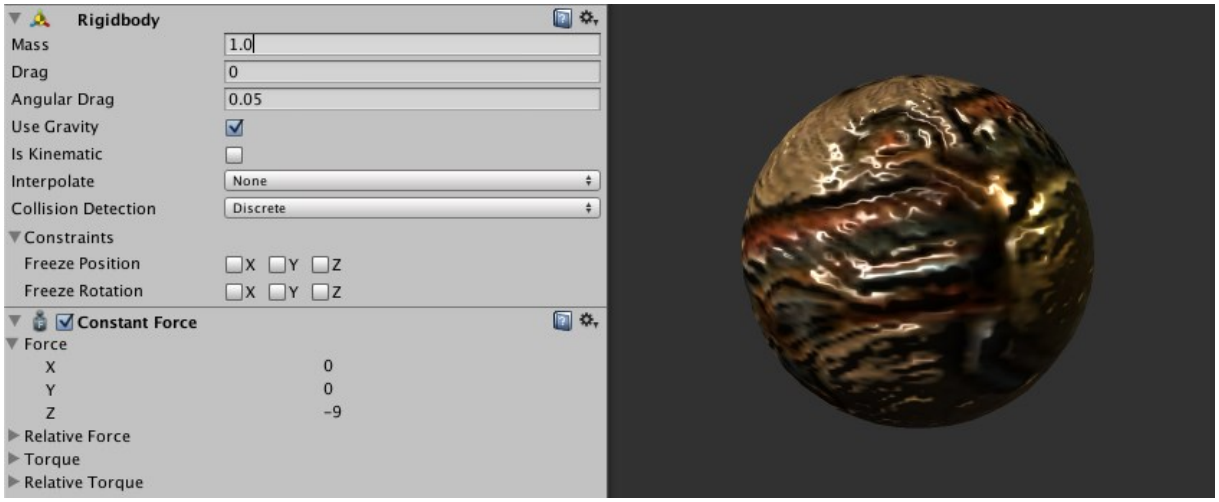
Primer 17: Funkciji, ki upravljata z meglo v drugem igralnem prostoru, nahajata v skripti *BuildWorld.cs*.

Slika 37: Različne barve megle v drugem igralnem prostoru.

4.2.7 Upravljanje s fizikalnimi silami

Unity ima vgrajen napreden fizikalni pogon NVIDIA PhysX [9]. Preko njega lahko nadziramo fizikalne sile na različne načine. Določene fizikalne pojave lahko vodimo za vsak posamezen objekt, druge pa na nivoju igralnega prostora.

Če želimo, da na posamezen objekt delujejo fizikalne sile, moramo objektu dodati komponento Rigidbody. Preko te komponente lahko v urejevalniku vplivamo na objekt s fizikalnimi silami (Slika 38).

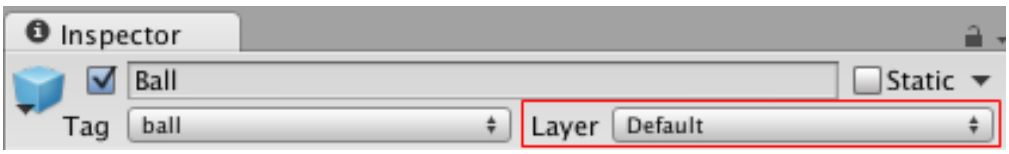


Slika 38: Primer nastavitev fizikalnih sil za objekt krogla. Objektu je dodana komponenta Rigidbody, v njej je nastavljena masa objekta na 1, upor (ang. angular drag) na 0.05, določeno je tudi, da naj na objekt deluje gravitacija. Dodana je tudi komponenta Constant Force, ki preko komponente Rigidbody določa silo (jakost sile v mojem primeru je -9 po osi z), ki naj deluje na objekt. Na ta način dosežemo, da se krogla kotali po osi z v nasprotni smeri, kot se giblje naše letalo.

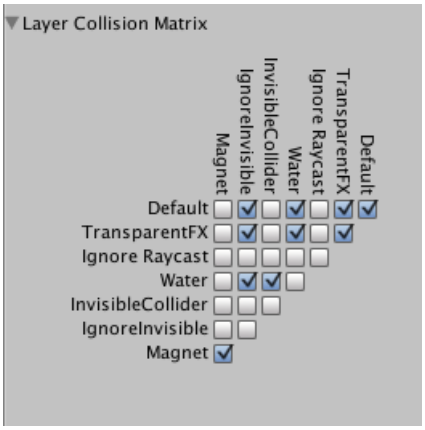
Fizikalne sile lahko določamo tudi na nivoju celotnega igralnega prostora. Na ta način določene lastnosti veljajo za vse objekte, ki so podvrženi fizikalnim pojavom (vsi objekti s komponento RigidBody) znotraj nekega igralnega prostora.

Plasti za zaznavanje trkov

Vsak objekt spada v določeno plast za zaznavanje trkov (Slika 39). Vse plasti pa sestavljajo tako imenovano matriko trkov plasti (ang. Layer Collision Matrix, Slika 40). Preko te matrike določimo zaznavanje plasti med seboj. Pri razvoju iger je v določenih primerih namreč praktično, da določeni objekti ne zaznavajo trkov z nekaterimi drugimi objekti. Na primer, za kroglo s slike 38 je pomembno, da ne zaznava trkov z zlatniki (se kotali skozi njih), zaznava pa trke z letalom.



Slika 39: Vsak objekt ima določeno (fizikalno) plast.

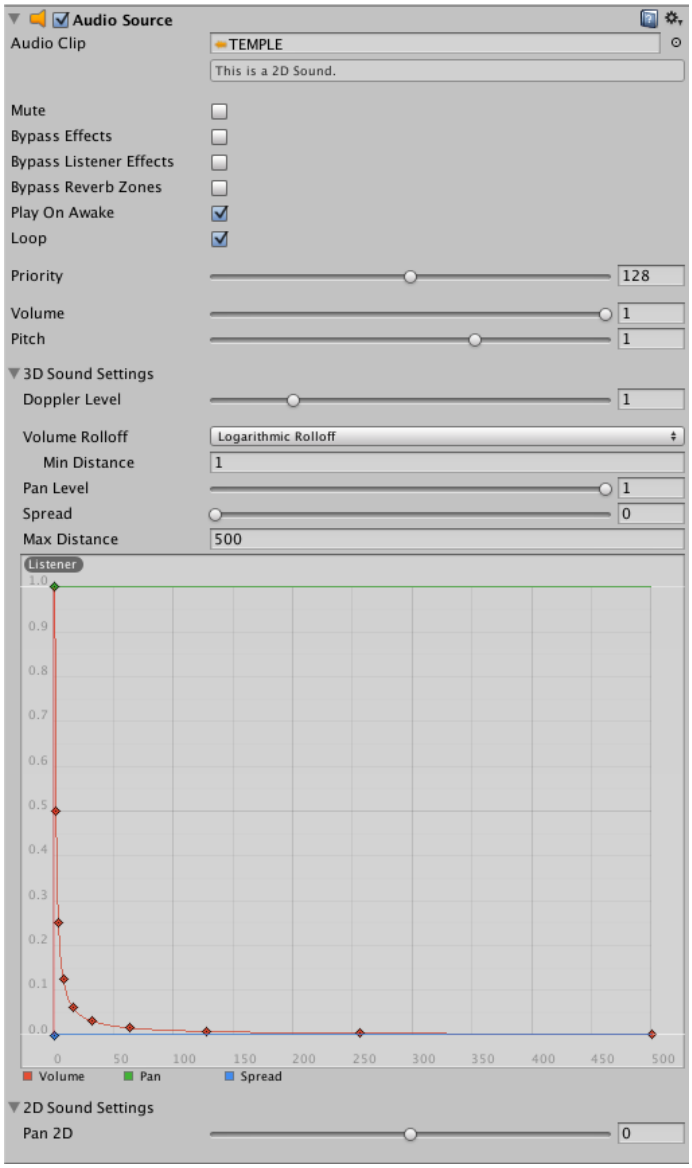


Slika 40: Z matriko zaznavanja trkov določimo katere plasti se naj zaznavajo med seboj.

4.2.8 Upravljanje z zvoki

Za boljšo izkušnjo pri igranju so poleg vizualnih učinkov zelo pomembni tudi zvočni učinki. V svoji igri uporabljam mnogo različnih zvokov, delijo se na osnovne zvoke, ki se pojavljajo ob interakciji z objekti v igralnem prostoru in na zvoke v ozadju (ang. ambient sounds). Osnovni zvoki so na primer zvok pri izstrelitvi rakete, različni zvoki eksplozij, zvok za klik na gumb in podobno. Zvok v ozadju je namenjen predvsem ustvarjanju vzdušja v igri in se vrti ciklično. V mojem primeru uporabljam za vsak igralni prostor po eno skladbo, ki se ciklično vrti v ozadju. V prvem igralnem polju je skladba mirna, v drugem pa je hitra in ustvarja občutek nevarnosti. Obe skladbi sta brezplačno dostopni na spletu [10], [11].

Tako kot večino ostalih elementov, lahko tudi zvok upravljamo preko kode ali pa preko urejevalnika Unity. Osnovne zvoke "prožim" večinoma preko kode (Primer 3, Primer 16) ob raznih dogodkih (pritisk na gumb, pobiranje zlatnika, eksplozija rakete...) s funkcijo *void PlayOneShot(AudioClip clip, float volume = 1.0F)* ali pa *void PlayClipAtPoint(AudioClip clip, Vector3 position, float volume = 1.0F).* Zvoke za ozadje pa upravljam preko urejevalnika Unity (Slika 41).



Slika 41: Primer nastavitev zvoka [13] za ozadje v prvem igralnem polju. Pomembni parametri za ta primer so "Play On Awake", kar pomeni, da se zvočni posnetek začne predvajati takoj, ko se naloži igralni prostor. Nastavitev "Loop" pomeni, da se naj zvočni posnetek predvaja ciklično.

4.2.9 Odstranjevanje objektov iz igralnega prostora

Ko letalo objekte obvozi, postanejo ti objekti nepotrebni in jih je treba odstraniti iz igralnega prostora, saj po nepotrebnem zasedajo pomnilnik in cikle procesorja. Za odstranjevanje teh objektov skrbi funkcija *DestroyObjects()* znotraj skripte *BuildWorld.cs*. Delovanje funkcije je preprosto: ciklično pregleduje vse objekte znotraj igralnega prostora, če se ti objekti nahajajo na določeni razdalji za letalom jih izbriše iz prostora (Primer 18).

```
IEnumerator DestroyObjects()
{
    while (true)
    {
        GameObject[] obj =
                GameObject.FindObjectsOfType(typeof(GameObject));

        foreach (GameObject g in obj)
        {
            /* Če se objekt nahaja 100 ali več enot po osi z za letalom,
            ga izbriši. */
            if (g.transform.position.z + 100 < Player.position.z)
            {
                Destroy(g);

                /* Počakaj 5 stotink sekunde med izbrisom posameznih
                objektov. Funkcija Destroy() je namreč procesorsko
                zahtevna. */
                yield return new WaitForSeconds(0.05f);
            }
        }

        yield return new WaitForSeconds(2);
    }
}
```

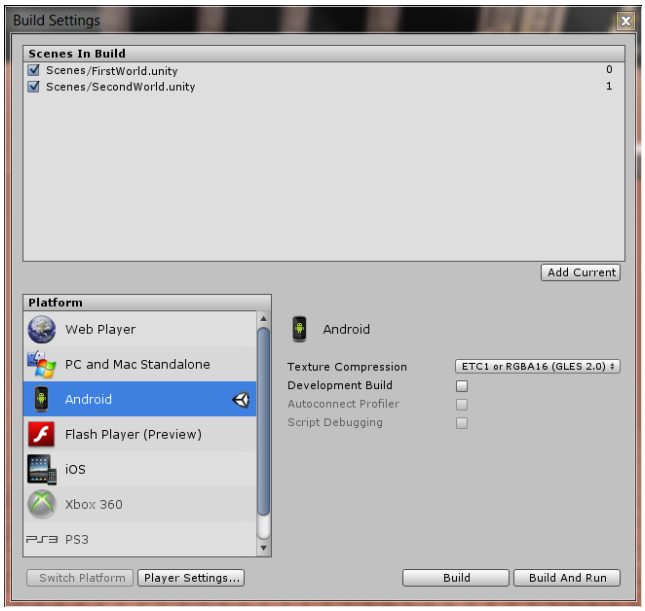
Primer 18: Funkcija, ki briše nepotrebne objekte iz igralnega prostora.

4.3 Priprava igre za objavo na portalu Google play

Ko je igra razvita, jo moramo pripraviti za objavo na portalu Google Play. Vso potrebno delo se opravi znotraj urejevalnika Unity.

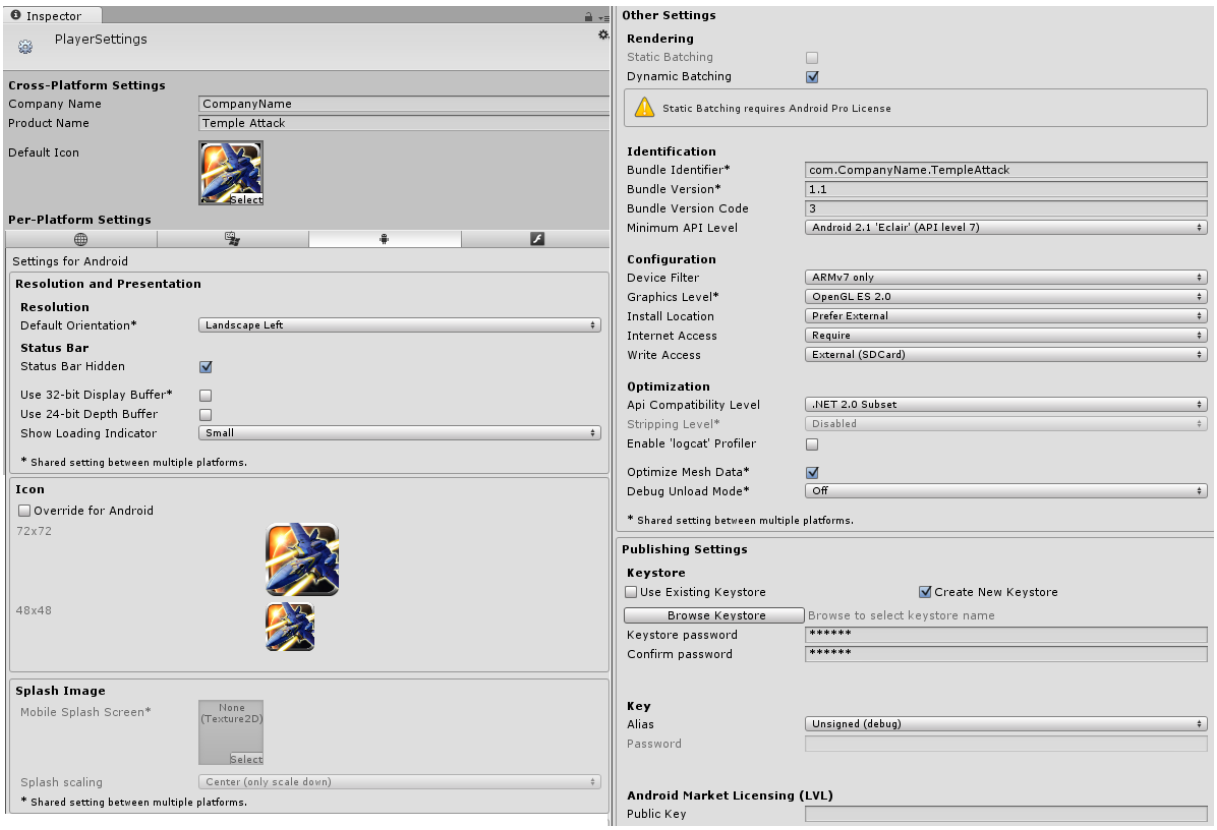
Glavni parametri, ki jih moramo določiti so (Slika 42, Slika 43):

- Vrsto kompresije za texture, v mojem primeru je to kompresija ETC1 oziroma RGBA16 (Slika 42).



Slika 42: Izbor ciljnega operacijskega sistema, izbor igralnih prostorov in vrsta kompresije tekstur.

- Ime podjetja, pod katerim bo igra prikazana na portalu.
- Ime igre oziroma aplikacije.
- Dodati moramo ikono.
- Določiti je potrebno omogočene orientacije mobilne naprave.
- Dodati prikazno sliko (ang. splash screen). To je omogočeno le s profesionalno licenco za Unity, drugače se na tem mestu samodejno pojavi logotip od orodja Unity.
- Določiti na katerih Android napravah bo igra delovala. To določimo posredno z določitvijo minimalne podprte verzije operacijskega sistema Android, npr. Android 2.1 Eclair (API level 7) in določitvijo podprte strojne opreme (npr. ARMv7 only).
- Dodati moramo tudi varnostni ključ, ki ga lahko generiramo kar znotraj urejevalnika Unity in pa javni ključ, ki ga dobimo na portalu Google Play.



Slika 43: Vse potrebne nastavitve [12] pred izgradnjo igre in objavo na Google Play portalu.

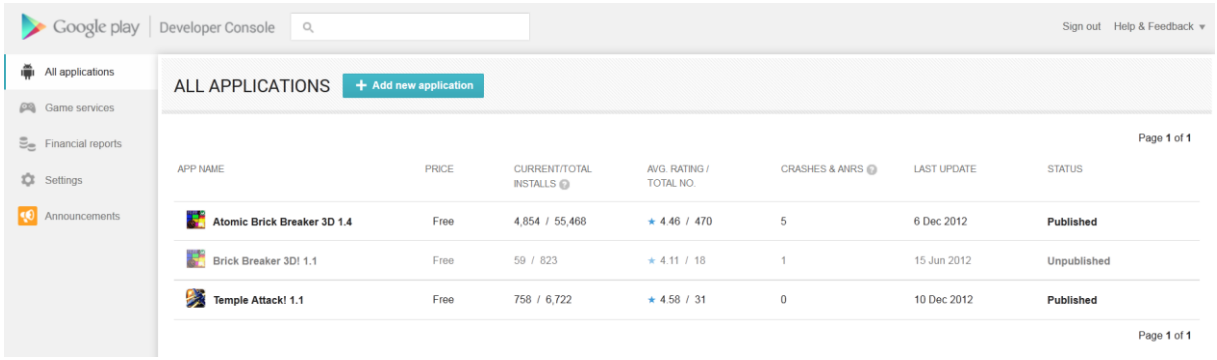
Ko ustrezno izpolnimo vse zahtevane vnose, lahko zgradimo aplikacijo formata .apk in jo naložimo na portal.

5 OBJAVA IGRE NA PORTALU GOOGLE PLAY IN REZULTATI

Pred objavo igre je potrebno odpreti račun na portalu Google Play. Obstajata dve vrsti računov: prvi se imenuje razvijalski račun, drugi pa trgovalni račun. Preko razvijalskega računa lahko objavljamo le brezplačne aplikacije, prav tako ne omogoča nakupov znotraj aplikacij (ang. IAP ali in app purchases) in naročnin (primer: mesečna naročnina na novice). V Sloveniji še ni možno odpreti trgovalskega računa in tudi nikjer ne obstaja podatek kdaj bo to možno. Zaradi tega sem lahko odprl le razvijalski račun, ob odprtju je treba plačati znesek 25$.

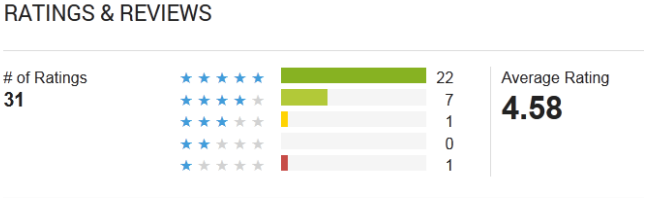
Podrobna dokumentacija za odpiranje Google Play računov je dostopna na spletu [14].

Google Play portal uporabniku tudi omogoča natančen pregled različnih statistik. V glavnem meniju so zbrani osnovni podatki o aplikacijah. Na svojem računu imam trenutno dve delujoči igri, obe sta brezplačni (Slika 44).



Slika 44: Glavni meni portala Google play. Tukaj je vidno število vseh inštalacij posameznih iger, število trenutno aktivnih inštalacij ter povprečna ocena igre s strani uporabnikov. V tabeli je vidno tudi kolikokrat se je aplikacija sesula in zmrznila, datum zadnje posodobitve na portalu, ter ali je igra trenutno objavljena ali ne.

Do dne 23.11.2013 si je igro preneslo s portala na svoje naprave 6722 uporabnikov. Nobeden od njih še ni prijavil, da bi se igra na napravi sesula ali zmrznila. Povprečna ocena igre s strani uporabnikov je 4,58 od 5 možnih (Slika 45).

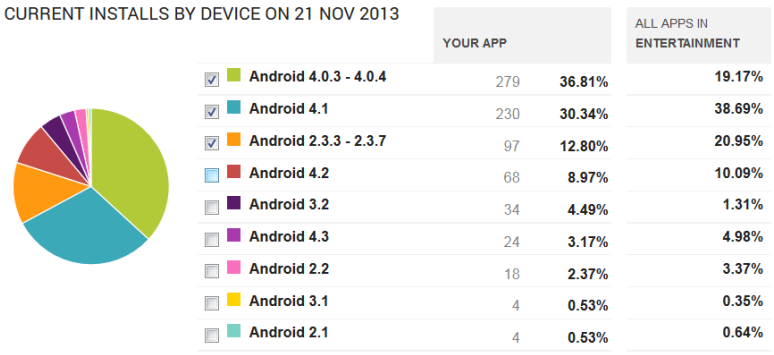


Slika 45: Ocene uporabnikov z Google play portala.

Na portalu lahko tudi vidimo katero verzijo operacijskega sistema imajo naloženo naprave uporabnikov. Najpogostejše so verzije Android 4.0.3 in 4.0.4, starejše verzije pa že krepko

zaostajajo. Iz tega lahko sklepamo, da večina Android uporabnikov dokaj redno nadgrajuje operacijski sistem (Slika 46).

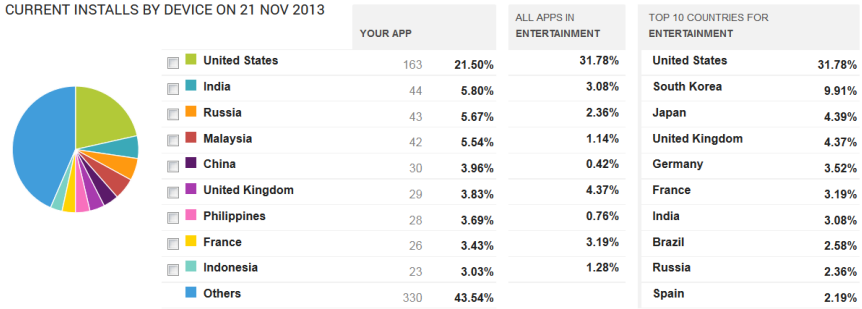
Pregled statistike po verzijah operacijskega sistema Android



Slika 46: Prvi stolpec opisuje verzijo operacijskega sistema Android, drugi stolpec pa delež uporabnikov, ki imajo trenutno naloženo igro na svojih napravah. Tretji stolpec prikazuje povprečni delež verzij operacijskega sistema glede na vse aplikacije na portalu, ki spadajo pod rubriko zabava.

Pregled uporabnikov po državah

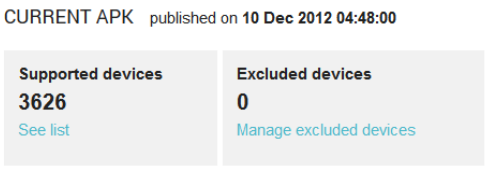
Največ aktivnih uporabnikov ima igra v ZDA (21,5 %), sledijo Indija s 5,8% in Rusija s 5,67% (Slika 47).



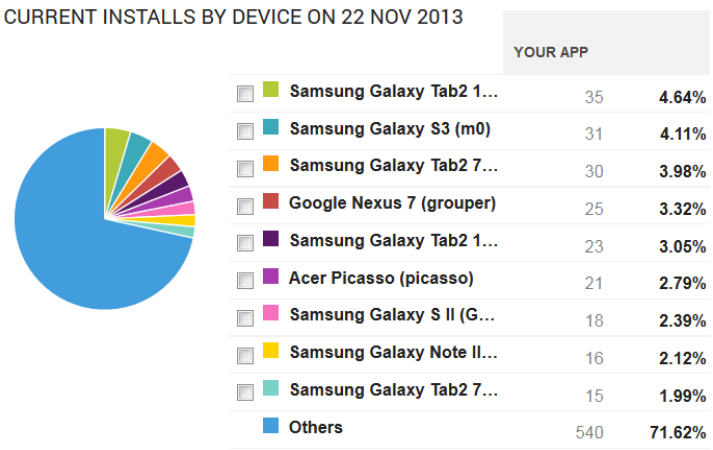
Slika 47: Število in delež aktivnih uporabnikov glede na državo. Tretji stolpec prikazuje kako je delež aktivnih uporabnikov porazdeljen glede na vse aplikacije iz kategorije zabava, ki so dostopne na portalu Google play. V četrtem stolpcu so po padajočem vrstnem redu porazdeljene države, kjer je kategorija zabava najpogostejša.

Pregled statistike po uporabljenih mobilnih napravah

Poleg tega, da lahko v urejevalniku Unity določimo zahtevano strojno opremo in verzijo operacijskega sistema (Slika 43), lahko na portalu Google Play natančno omejimo na katerih napravah naj bo aplikacija omogočena. Če ima uporabnik napravo, ki smo jo izključili iz seznama podprtih naprav, si aplikacije ne more namestiti. Ni pa nujno, da bo aplikacija delovala, četudi je naprava na seznamu delujočih naprav. Za mojo igro nisem na portalu Google Play izvzemal nobenih naprav (omejitve veljajo le iz urejevalnika Unity), zato je podprtih kar 3626 naprav (Slika 48), po deležu pa močno prevladujejo naprave podjetja Samsung (Slika 49).



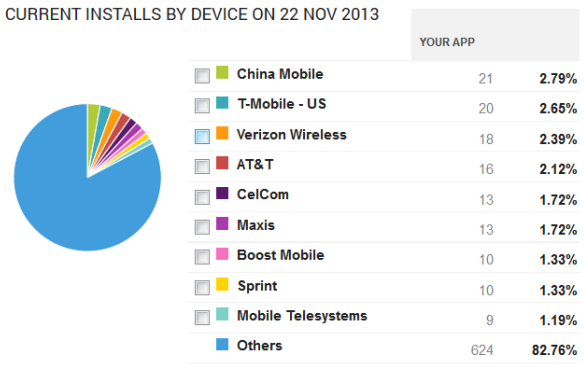
Slika 48: Število podprtih naprav. To so naprave, ki imajo procesor ARMv7 in podpirajo najmanj verzijo operacijskega sistema Android 2.1. Omejitev izhaja le iz nastavitev v urejevalniku Unity (Slika 43).



Slika 49: Deleži mobilnih naprav z aktivno namestitvijo igre.

Pregled statistike po ponudnikih mobilnega omrežja

Preko portala je tudi možno določiti v katerih državah naj bo igra dostopna, možno je tudi blokirati nekatere ponudnike mobilnega omrežja. Moja igra je dostopna po celem svetu, blokiral tudi nisem nobenega ponudnika mobilnih omrežij. Največ aktivnih uporabnikov igre spada pod China Mobile, sledita pa mu T-Mobile in Verizon Wireless (Slika 50).



Slika 50: Porazdelitev uporabnikov igre med ponudniki mobilnega omrežja.

6 ZAKLJUČKI

Glavni cilj diplomskega dela je bil prikazati razvoj 3D igre, ki deluje na pametnih telefonih in na tabličnih računalnikih z operacijskim sistemom Android. Za razvoj sem uporabil razvojno orodje Unity, na kratko sem tudi opisal njegove glavne značilnosti in jih prikazal na primerih iz prakse. Glavne naloge pri razvoju 3D igre bi lahko razdelil na: programiranje, delo z zvoki, delo s 2D teksturami, ter delo s 3D modeli. In pa seveda povezovanje vseh teh elementov v zaključeno celoto.

Igro sem tudi objavil na Google Play portalu in kasneje še na applovem portalu iTunes. Uporabniki so igro ocenili na obeh portalih dokaj visoko, zaradi tega imam tudi zagon za nadaljnji razvoj te igre. Ker v Sloveniji ni možno odpreti trgovalskega računa na portalu Google Play, se bom v prihodnosti osredotočil predvsem na operacijski sistem iOS. K sodelovanju bom tudi poskušal pridobiti 3D oblikovalca, ki bo izboljšal in poenotil vizualno podobo igre. Ko bo igra še bolj izpopolnjena pa se bom posvetil oglaševanju ali pa iskanju založnika igre. Največji problem neodvisnih razvijalcev je ravno preboj na vidna mesta. Ko igro objavimo na portalu med več kot milijon ostalimi aplikacijami, se le ta v množici hitro izgubi in zaradi tega doseže nizko število namestitev.

Poleg opisane igre v diplomski nalogi, imam na portalu še eno igro, z imenom Atomic Brick Breaker. Čeprav je ta igra po mojem mnenju slabše kakovosti, je dosegla veliko večjo število namestitev, in sicer 55.466 (Slika 44). Vzrok za takšno razliko mi ni znan, saj nobene od teh iger še nisem nikjer oglaševal.

Glede na to, da sem igro v celoti ustvaril sem, sem z izdelkom, še posebej z ocenami uporabnikov zelo zadovoljen. Nisem pa še zadovoljen z številom namestitev, povečanje tega števila mi bo v prihodnosti predstavljalo največji izziv.

Igra je brezplačno dostopna na naslovih:

- Verzija za Android: http://goo.gl/f55ghl
- Verzija za iOS: http://goo.gl/AXtuYQ

7 VIRI IN LITERATURA

[1] (2014) Članek o številu prenosov in številu vseh aplikacij na portalu Google Play, dostopen na:
http://www.theverge.com/2013/7/24/4553010/google-50-billion-android-app-downloads-1m-apps-available

[2] (2014) Audacity, program za delo z zvokovnimi datotekami. Dokumentacija in sam program, dostopno na:
http://audacity.sourceforge.net/about/

[3] (2014) Paint.NET, program z dokumentacijo za delo z 2D teksturami, dostopen na:
http://www.getpaint.net/index.html

[4] (2014) MonoDevelop, razvojno orodje in dokumentacija, dostopen na:
http://monodevelop.com/

[5] (2014) Blender, program za oblikovanje 3D modelov z dokumentacijo, dostopen na:
http://www.blender.org

[6] Patrick Felicia, Getting Started with Unity, Packt Publishing, 2013

[7] (2014) Unity GameObjects, dokumentacija:
http://docs.unity3d.com/Documentation/Manual/GameObjects.html

[8] (2014) Unity PlayerPrefs, dokumentacija:
http://docs.unity3d.com/Documentation/ScriptReference/PlayerPrefs.html

[9] (2014) Fizikalni pogon NVIDIA PhysX, dokumentacija dostopna na:

https://developer.nvidia.com/physx

[10](2014) Skladba, ki se vrti v ozadju prvega igralnega prostora, dostopna na:
http://www.flashkit.com/soundfx/Ambience/TEMPLE-Adam_Goh-7394/index.php

[11] (2014) Skladba, ki se vrti v ozadju drugega igralnega prostora, dostopna na:
http://www.nosoapradio.us/

[12] (2014) Unity Player Settings, dokumentacija dostopna na:
http://docs.unity3d.com/Documentation/Manual/class-PlayerSettings.html

[13] (2014) Unity Audio Source, dokumentacija dospona na:
http://docs.unity3d.com/Documentation/ScriptReference/AudioSource.html

[14] (2014) Odpiranje Google play razvijalskega računa, dokumentacija dostopna na:
http://developer.android.com/distribute/googleplay/publish/register.html

[15] (2014) Unity GUI Style, dokumentacija dostopna na:
http://docs.unity3d.com/Documentation/Components/class-GUIStyle.html

[16] (2014) Nastavitve grafike v orodju Unity, dokumentacija dostopna na:
http://docs.unity3d.com/Documentation/Components/class-QualitySettings.html

[17] (2014) Spletni generator tekstur. Dostopno na:
http://flamingtext.com/

SEZNAM SLIK: